Hybrid Publicly Verifiable Computation

James Alderman[*], Christian Janson, Carlos Cid[†], and Jason Crampton

Information Security Group, Royal Holloway, University of London
Egham, Surrey, TW20 0EX, United Kingdom
{James.Alderman.2011, Christian.Janson.2012}@live.rhul.ac.uk
{Carlos.Cid, Jason.Crampton}@rhul.ac.uk

Abstract

Publicly Verifiable Outsourced Computation (PVC) allows weak devices to delegate computations to more powerful servers, and to verify the correctness of results. Delegation and verification rely only on public parameters, and thus PVC lends itself to large multi-user systems where entities need not be registered, yet in such settings the individual user requirements may be diverse. In this paper, we introduce *Hybrid PVC* (HPVC) which, with a single setup stage, provides a flexible solution to outsourced computation supporting standard PVC, the enforcement of access control policies restricting the servers that may evaluate a given computation, and a reversed model of PVC which we call *Verifiable Delegable Computation* (VDC) where data is held remotely by servers. We provide formal frameworks and constructions for such systems.

Keywords— Hybrid Publicly Verifiable Computation, Verifiable Delegable Computation, Dual-Policy Attribute-based Encryption, MapReduce, Access Control

1 Introduction

We consider the use of attribute-based encryption in the context of Verifiable Outsourced Computation (VC) [22, 17, 37, 20, 15, 11, 29] to achieve public verifiability and delegation. The trend towards cloud computing means that there is a growing trust dependency on remote servers and the functionality that they provide. *Publicly Verifiable Computation* (PVC) allows *any* entity to use public information to delegate or verify computations, and lends itself to large multi-user systems (as delegators need not be individually registered). However, in such a system, the individual user requirements may be diverse. In this work, we provide a flexible solution, called *Hybrid PVC*, that enables multiple modes of operation with a single set of system parameters. In particular, we capture:

- *Revocable Publicly Verifiable Outsourced Computation* (RPVC) [2] where servers are certified to evaluate certain functions and delegators use public parameters to send input

[*]The first author acknowledges support from BAE Systems Advanced Technology Centre under a CASE Award.

[†]This research was partially sponsored by US Army Research laboratory and the UK Ministry of Defence under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defence, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

data to a server and to verify correctness of the result. Servers may try to return incorrect results to persuade verifiers of incorrect information or to avoid using computational resources. Misbehaving servers can be detected and prevented from performing further evaluations;

- an extension of *Publicly Verifiable Computation with Access Control* (PVC-AC) [3] to enforce restrictions on which servers may perform a given computation (based on factors such as sensitivity of the input data, physical server location etc.). This is motivated by the possibility that, within a large system, outsourced computations may be distributed amongst a pool of available servers that are not individually authenticated by the delegator and so there may be less control over access to computational data. Prior work required all entities to be registered in the system (including delegators) but we achieve a fully public system where only servers need be registered (as required in usual Publicly Verifiable Outsourced Computation anyway);

- a reversal of the usual model for Publicly Verifiable Computation where remote servers make available a static database over which any delegator may request computations (or queries) to be performed. This architecture, which we call *Verifiable Delegable Computation* (VDC), can naturally be applied to problems such as MapReduce and verifiable queries on remote databases. The *efficiency* requirement for this model differs from the classical VC setting; outsourcing a computation is no longer merely an attempt to gain efficiency since the delegator is never in possession of the input data and cannot execute the computation himself. Our solution achieves constant time public verification and the communication costs to delegate computations depends on the function F, while the size of the response depends on the size of $F(x)$ and *not* on the size of x which may be large, particularly when querying remote databases.

We begin by recapping related work and provide a formal definition and construction for our third mode of operation, VDC, built from a novel use of Ciphertext-policy Attribute-based Encryption (CP-ABE). This construction will inform that of HPVC in Sect. 3. We then define HPVC and see that it provides a natural application for Dual-Policy Attribute-based Encryption (DP-ABE) [7] which conjunctively combines both KP- and CP-ABE. We introduce a new primitive called *Revocable Dual-policy Attribute-based Encryption (DP-ABE)* which is a building block of our HPVC construction which we introduce in Sect. 3. Full details, construction and security proof for this primitive can be found in App. B. DP-ABE has previously attracted relatively little attention in the literature, which we believe to be because applications for the primitive are less obvious than for the single-policy ABE schemes which are often used to cryptographically enforce access control policies (where objects are encrypted and can only be read by users holding keys satisfying some decryption policy). It is unclear that the policies enforced by DP-ABE are natural choices for access control. Thus an interesting side-effect of this work is to show that additional applications for DP-ABE exist. Finally, in Sect. 4, we observe that the above use of DP-ABE in HPVC does not use the full power of DP-ABE, instead using DP-ABE to separately enforce KP- and CP policies. We show that by using both forms of policy *simultaneously* we can enforce access control policies on the servers that can perform a computation. Furthermore, we provide a discussion about access control on the delegator in App. C. Finally in App. D we briefly present another application of DP-ABE where we demonstrate how to use DP-ABE in order to achieve a form of entity authentication and authenticated key agreement protocol.

1.1 Related Work

Verifiable computation [22] may be seen as a protocol between two polynomial-time parties: a (weak) client C and a server S which results in the provably correct computation of $F(x)$ by the server for the client's choice of x. There may be a computationally expensive setup stage (amortised over multiple computations of F) but other operations should be efficient for the client. Some prior work used garbled circuits with fully homomorphic encryption [22, 17] or targeted specific functions [11, 20, 15]. Chung et al. [18] introduced *memory delegation* which is close to our notion of VDC; here, the client uploads his memory to a server who can update and compute a function F over the entire memory. Backes et al. [9] considers a client that outsources a large amount of data and requests computations on a data portion. The client can efficiently verify the correctness of the result without holding the input data. Most work in this realm of outsourced data requires the client to know the data to verify e.g. in SNARG-based approaches [23, 10, 13] and signatures of correct computation [28]. Apon et al. [4] propose a notion of *verifiable oblivious storage* to ensure data confidentiality, access pattern privacy, integrity and freshness of data accesses of data. Work from the realm of authenticated data lends itself to the concept of verifiable computations over outsourced data, albeit for specific functions only. Backes et al. [8] consider computations over outsourced data based on privacy-preserving proofs over authenticated data outsourced by a trusted client. Similar results were presented in [34] using public logs. It is notable that the work by [8] and [13] also achieve the notion of public verifiability. In independent and concurrent work, Shi et al. [33] considered using DP-ABE to combine keyword search on encrypted data with the enforcement of an access control policy.

Parno et al. [29] introduced *Publicly Verifiable Computation* (PVC) where multiple clients can outsource and verify computations. Alderman et al. [2] extended this to include a trusted Key Distribution Centre (KDC), multiple servers and to revoke misbehaving servers. Informally, the KDC acts as the root of trust to generate public parameters and to issue personalized secret keys, evaluation keys for servers, and public delegation information. To outsource the evaluation of $F(x)$, a delegator C sends an encoded input $\sigma_{F(x)}$ to a server S, and publishes verification tokens for the computation. S uses an evaluation key for F to produce an encoded output $\theta_{F(x)}$ which any entity can blindly verify correctness of using the verification token. The verifier may not learn the value of $F(x)$ if not in possession of the retrieval key. If S cheated they may report S to the KDC for revocation, otherwise the retrieval key $RK_{F(x)}$ can be used to recover $F(x)$.

2 Verifiable Delegable Computation

In prior models of PVC [29, 2], delegators owned the input data and outsourced computations on that data to a more powerful server. It is often the case however that the server owns a large database and makes portions of it available to be queried. This fits with the traditional client-server model and is also similar to other areas of verifiable computation such as memory delegation. We refer to this model as Verifiable Delegable Computation (VDC) and believe it can applied to many practical problems. For example:

- **MapReduce** [19] (or Hadoop [36]) is a framework for parallel processing of large computations where a set of worker nodes each compute subproblems on portions of the data and report to a manager who combines the results. VDC enables verifiable MapReduce such that only *valid* results are combined. The manager acts as the Key Distribution Center (KDC) to distribute evaluation keys for partitions of the data to workers. He can then request multiple sub-problems to be solved over this data partitioning.

- **Verifiable queries on remote databases.** Servers may also act as remote database providers and register with a KDC to provide a verifiable querying service. Any delegator

(a) PVC (b) VDC

Figure 1: Comparison between PVC and VDC

may use public information to query *any* function (within the family allowed by the VDC scheme) on these databases. Data is remotely stored and delegators see nothing more than the results of queries which they are assured are correct.

Parno et al. [29] showed that KP-ABE provides a proof that a Boolean circuit was satisfied in the PVC setting. In a similar way, we believe CP-ABE is a natural choice to provide VDC functionality. Data remains statically stored at the server and may be embedded in a server's secret key, whilst the computation of many different functions can be requested by creating ciphertexts using *only* public information. In the PVC setting, efficiency is a strong requirement – it must be cheaper to outsource and verify a computation than to perform it locally. With VDC, we do not require such stringent efficiency conditions since, without holding the input data, delegators necessarily must outsource the computation; outsourcing is no longer merely an attempt to gain efficiency but to gain functionality. Despite this, CP-ABE behaves reasonably well in this setting. The outsourcing of a computation does, unfortunately, require work comparable to performing the computation itself, but verification is constant time and very efficient and we achieve efficient communication costs in that the size of results is $O(|F(x)|)$ and does not depend on the size of the data (unlike some naive methods). Future work should concentrate on reducing the cost of outsourcing.

2.1 Specification

Let there be n servers S_i, each holding one input x_i, $1 \leqslant i \leqslant n$. Informally, a VDC scheme for a family of functions $\mathcal{F}$ begins with a Key Distribution Center (KDC) (e.g. a trusted third party or a delegator) running Setup to produce public parameters and a master secret key. The KDC also registers each server S_i to provide a private signing key for SK_{S_i}, and publishes a public delegation key PK_F for each function of interest F. Each server, S_i registers their interest in performing work on data x_i by requesting a (single) evaluation key EK_{x_i} from the KDC in the Certify stage. They also provide a list of functions, $\mathcal{F}_i \subseteq \mathcal{F}$, that they are willing to evaluate[1]. The KDC maintains a public list L_{Reg} that, for each server, lists the functions they are willing to compute and a unique description $l(x_i)$ of their data. The unique label means that delegators may choose servers and data with *only* knowledge of this label (e.g. a database name) and need not know the data itself (we assume that the KDC assigns or verifies this label). The ProbGen algorithm requests the computation of F. The delegator chooses a server S_i with data label $l(x_i)$ from the list L_{Reg}. In the MapReduce example, ProbGen would be run once per worker for the same function F. ProbGen generates an encoded input $\sigma_{F(x_i)}$, verification key $VK_{F(x_i)}$ and output retrieval key $RK_{F(x_i)}$. A server S_i uses their evaluation key EK_{x_i} to compute $\theta_{F(x_i)}$ encoding $F(x_i)$.

[1]Each server S_i may specify the functions $\mathcal{F}_i \subseteq \mathcal{F}$ that they are willing to perform. In settings such as MapReduce, this may be $\mathcal{F}$ since the delegator is the data owner and distributes input data. However when servers are remote data providers, they themselves own the input data and should specify what that data be used for.

Verification is divided into two algorithms. We consider public verifiability where *any* party (including an adversary) may verify a result. An example motivation is MapReduce in a grid computing environment where some management software ensures results are correct before returning them to the user. Blind verification is performed by *any* user using a verification key $VK_{F(x_i)}$ to verify correctness. It generates a retrieval token $RT_{F(x_i)}$ for valid outputs (or $\perp$ otherwise) but does *not* reveal the actual output value. Finally, $\mathsf{Retrieve}$ takes a (non-$\perp$) token $RT_{F(x_i)}$ and the output retrieval key $RK_{F(x_i)}$ to reveal the final result $y_{F(x_i)} = F(x_i)$. More formally:

Definition 1. *A* Publicly Verifiable Delegable Computation (VDC) scheme *comprises the following algorithms*[2]*:*

1. $(PP, MK) \leftarrow \mathsf{Setup}(1^\kappa)$
2. $PK_F \leftarrow \mathsf{FnInit}(F, MK, PP)$
3. $SK_{S_i} \leftarrow \mathsf{Register}(S_i, MK, PP)$
4. $EK_{x_i,S_i} \leftarrow \mathsf{Certify}(S_i, x_i, l(x_i), \mathcal{F}_i, MK, PP)$
5. $(\sigma_{F(x_i)}, VK_{F(x_i)}, RK_{F(x_i)}) \leftarrow \mathsf{ProbGen}(F, l(x_i), PK_F, PP)$
6. $\theta_{F(x_i)} \leftarrow \mathsf{Compute}(\sigma_{F(x_i)}, EK_{x_i,S_i}, SK_{S_i}, PP)$
7. $y_{F(x_i)} \leftarrow \mathsf{Verify}(\theta_{F(x_i)}, RT_{F(x_i)}, VK_{F(x_i)}, RK_{F(x_i)}, PP)$:
   - $(RT_{F(x_i)} \text{ or } \perp) \leftarrow \mathsf{BVerif}(\theta_{F(x_i)}, VK_{F(x_i)}, PP)$
   - $y_{F(x_i)} \leftarrow \mathsf{Retrieve}(RT_{F(x_i)}, VK_{F(x_i)}, RK_{F(x_i)}, PP)$

We do not consider revocation here but observe that an indirectly revocable CP-ABE scheme could be employed in a similar fashion to [2]. Additionally, the hybrid scheme in Sect. 3 can revoke policies in decryption keys – revoking a "dummy policy" disables all evaluation keys for the entity (even for VDC).

We also assume a mechanism (e.g. tagged messages) linking messages related to a single computation. Note that the server must provide its input data in order to be certified by the trusted KDC. In some settings, the data may be partitioned beforehand by the KDC and distributed to the servers along with their keys. For example, if the manager of an organization divides a database according to a separation of duty policy and allows particular verifiable queries. In other settings, the server may have to trust the KDC with its data (but not the delegators). By restricting the issuing of keys, servers may not use a key for data they do not own (or indeed a subset of the data). Future work will attempt to relax this requirement.

A VDC scheme is *correct* if verification almost certainly succeeds when all algorithms are run honestly. A more formal definition follows:

Definition 2. *A Verifiable Delegable Computation system is* correct *for a family of functions $\mathcal{F}$ if for all functions $F \in \mathcal{F}$, servers S_i and inputs x_i where* $\mathrm{negl}(\cdot)$ *is a negligible function of*

[2]We retain the algorithm names from prior PVC schemes for consistency.

its input:

$$\begin{aligned}
&\Pr[(PP, MK) \leftarrow \mathsf{Setup}(1^\kappa), PK_F \leftarrow \mathsf{FnInit}(F, MK, PP),\\
&\quad SK_{S_i} \leftarrow \mathsf{Register}(S_i, MK, PP),\\
&\quad EK_{x_i,S_i} \leftarrow \mathsf{Certify}(S_i, x_i, l(x_i), \mathcal{F}_i, MK, PP),\\
&\quad (\sigma_{F(x_i)}, VK_{F(x_i)}, RK_{F(x_i)}) \leftarrow \mathsf{ProbGen}(F, l(x_i), PK_F, PP),\\
&\quad y_{F(x_i)} \leftarrow \mathsf{Verify}(\mathsf{Compute}(\sigma_{F(x_i)}, EK_{x_i,S_i}, SK_{S_i}, PP), VK_{F(x_i)}, RK_{F(x_i)}, PP)]\\
&\quad = 1 - \mathrm{negl}(\kappa).
\end{aligned}$$

2.2 Security Models

We now introduce several security models capturing different requirements of a VDC scheme. We will formalize these notions of security as a series of cryptographic games run by a challenger. The adversary against a particular function F is modelled as a PPT algorithm $\mathcal{A}$ run by a challenger with input parameters chosen to represent the knowledge of a real attacker as well as the security parameter κ. The adversary algorithm may maintain state and be multi-stage (i.e. be called several times by the challenger, with different input parameters) and we overload the notation by calling each of these adversary algorithms $\mathcal{A}$. This represents the adversary performing tasks at different points during the execution of the system, and we assume that the adversary may maintain a state storing any knowledge it gains during each phase (we do not provide the state as an input or output of the adversary for ease of notation). The notation $\mathcal{A}^{\mathcal{O}}$ denotes the adversary $\mathcal{A}$ being provided with oracle access to the following functions: $\mathsf{FnInit}(\cdot, MK, PP)$, $\mathsf{Register}(\cdot, MK, PP)$, $\mathsf{Certify}(\cdot, (\cdot, \cdot), \cdot, \cdot, \cdot MK, PP)$ and $\mathsf{Revoke}(\cdot, (\cdot, \cdot), (\cdot, \cdot), MK, PP)$. This means that the adversary can query (multiple times) the challenger for any of these functions with the adversary's choice of values for parameters represented with a dot above. This models information the adversary could learn from observing a functioning system or by acting like a legitimate client (or corrupting one) to request some functionality.

In particular for the VDC framework we consider security against *Public Verifiability* and *Revocation*.

2.2.1 Public Verifiability

We formalize security against Public Verifiability in the VDC framework. This notion accomodates that multiple servers should not be able to collude in order to gain an advantage in convincing *any* verifying party of an incorrect result.

Full Public Verifiability. This is captured in Game 1. The game begins with the challenger setting up the system and running FnInit to initialize the challenge function F. Then the adversary $\mathcal{A}$ is given the resulting public parameters and given oracle access to $\mathsf{FnInit}(\cdot, MK, PP)$, $\mathsf{Register}(\cdot, MK, PP)$, $\mathsf{Certify}(\cdot, (\cdot, \cdot), \cdot, \cdot, \cdot MK, PP)$ and $\mathsf{Revoke}(\cdot, (\cdot, \cdot), (\cdot, \cdot), MK, PP)$ as mentioned previously. All oracles simply run the relevant algorithm. Eventually, the adversary will finish this query phase and output challenge input and corresponding label $(x^\star, l(x^\star))$. Then the challenger forms a challenge by running $\mathsf{ProbGen}$ on this input and sends the resulting encoded input to $\mathcal{A}$. The adversary is provided with oracle access again and wins the game if it is able to produce an encoded output that verifies correctly but does not correspond to the actual result $F(x^\star)$.

Game 1 $\mathbf{Exp}_{\mathcal{A}}^{PubVerif}[\mathcal{VDC}, F, 1^\kappa]$:

1: $(PP, MK) \leftarrow \mathsf{Setup}(1^\kappa)$
2: $PK_F \leftarrow \mathsf{FnInit}(F, MK, PP)$
3: $(x^\star, l(x^\star)) \leftarrow \mathcal{A}^{\mathcal{O}}(PK_F, PP)$
4: $(\sigma_{F(x^\star)}, VK_{F(x^\star)}, RK_{F(x^\star)}) \leftarrow \mathsf{ProbGen}(F, l(x^\star), PK_F, PP)$
5: $\theta_{F(x^\star)} \leftarrow \mathcal{A}^{\mathcal{O}}(\sigma_{F(x^\star)}, VK_{F(x^\star)}, RK_{F(x^\star)}, PK_F, PP)$
6: $RT_{F(x^\star)} \leftarrow \mathsf{BVerif}(\theta_{F(x^\star)}, VK_{F(x^\star)}, PP)$
7: $y_{F(x^\star)} \leftarrow \mathsf{Retrieve}(RT_{F(x^\star)}, RK_{F(x^\star)}, PP)$
8: **if** $(y_{F(x^\star)} \neq \perp)$ **and** $(y_{F(x^\star)} \neq F(x^\star))$ **then return** 1
9: **else return** 0

Game 2 $\mathbf{Exp}_{\mathcal{A}}^{sPubVerif}[\mathcal{VDC}, F, 1^\kappa]$:

1: $(x_i^\star, l(x_i^\star)) \leftarrow \mathcal{A}(1^\kappa)$
2: $(PP, MK) \leftarrow \mathsf{Setup}(1^\kappa)$
3: $PK_F \leftarrow \mathsf{FnInit}(F, MK, PP)$
4: $(\sigma_{F(x_i^\star)}, VK_{F(x_i^\star)}, RK_{F(x_i^\star)}) \leftarrow \mathsf{ProbGen}(F, l(x_i^\star), PK_F, PP)$
5: $\theta_{F(x_i^\star)} \leftarrow \mathcal{A}^{\mathcal{O}}(\sigma_{F(x_i^\star)}, VK_{F(x_i^\star)}, RK_{F(x_i^\star)}, PK_F, PP)$
6: $RT_{F(x_i^\star)} \leftarrow \mathsf{BVerif}(\theta_{F(x_i^\star)}, VK_{F(x_i^\star)}, PP)$
7: $y_{F(x_i^\star)} \leftarrow \mathsf{Retrieve}(RT_{F(x_i^\star)}, RK_{F(x_i^\star)}, PP)$
8: **if** $(y_{F(x_i^\star)} \neq \perp)$ **and** $(y_{F(x_i^\star)} \neq F(x_i^\star))$ **then return** 1
9: **else return** 0

Selective Public Verifiability. In Game 2, we capture the security notion of *selective public verifiability*. This is a selective notion of security for a given challenge function F where, at the start of the game, the adversary chooses the challenge input data $x_i^\star$ with unique label $l(x_i^\star)$. The challenger then initializes the system, runs FnInit for the challenge function F to create the public delegation key, and runs $\mathsf{ProbGen}$ on $x_i^\star$. The adversary is given the resulting outputs, all public information as well as oracle access. Thus he may create delegation keys for any function in $\mathcal{F}$, simulate the corruption of other servers by registering them and certifying them for input data of his choice, and view details of other delegated computations. The adversary eventually outputs $\theta_{F(x_i^\star)}$ which it believes to be an incorrect result that will, nevertheless, be accepted by a verifier. The challenger runs the verification steps on this output and the adversary wins if verification succeeds yet the result is not $F(x_i^\star)$.

The requirement for a selective notion stems from the underlying CP-ABE primitive used in the construction: a selectively secure CP-ABE scheme will result in selective Public Verification for the VDC scheme, whilst a fully secure CP-ABE scheme will yield (in exactly the same fashion) a fully Public Verifiable VDC scheme. We focus here on the selective case purely to maintain consistency with our notion of HPVC which, using current primitives, is only selectively secure. However, it is straightforward to amend the proof to accommodate full security if desired.

Definition 3. *The* advantage *of an adversary $\mathcal{A}$ running in probabilistic polynomial time (PPT), making a polynomial number of queries q, where $\mathbf{X} \in \{PubVerif, sPubVerif\}$, is defined as:*

$$Adv_{\mathcal{A}}^{\mathbf{X}}(\mathcal{VDC}, F, 1^\kappa, q) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{\mathbf{X}}[\mathcal{VDC}, F, 1^\kappa] = 1].$$

A VDC scheme is secure *against Game $\mathbf{X}$ for a function F, if for all PPT adversaries $\mathcal{A}$, $Adv_{\mathcal{A}}^{\mathbf{X}}(\mathcal{VDC}, F, 1^\kappa, q) \leq \mathrm{negl}(\kappa)$, where $\mathrm{negl}(\cdot)$ is negligible in its input.*

2.2.2 Blind Verification

The notion of Blind Verification, captured in Game 3, ensures that a verifier without the output retrieval key $RK_{F(x)}$ cannot learn the value of $F(x)$ from the encoded output (this does not

Game 3 $\mathbf{Exp}_{\mathcal{A}}^{BVerif}[\mathcal{VDC}, F, 1^{\kappa}]$:

1: $(PP, MK) \leftarrow \mathsf{Setup}(1^{\kappa})$;
2: $PK_F \leftarrow \mathsf{FnInit}(F, MK, PP)$;
3: $x \xleftarrow{\$} \mathsf{Dom}(F)$;
4: $S_i \xleftarrow{\$} \mathcal{U}_{ID}$;
5: $SK_{S_i} \leftarrow \mathsf{Register}(S_i, MK, PP)$;
6: $EK_{x,S_i} \leftarrow \mathsf{Certify}(S_i, x, l(x), \{F\}, MK, PP)$;
7: $(\sigma_{F(x)}, VK_{F(x)}, RK_{F(x)}) \leftarrow \mathsf{ProbGen}(F, l(x), PK_F, PP)$;
8: $\theta_{F(x)} \leftarrow \mathsf{Compute}(\sigma_{F(x)}, EK_{x,S_i}, SK_{S_i}, PP)$;
9: $\hat{y} \leftarrow \mathcal{A}^{\mathcal{O},\mathsf{Retrieve}}(\theta_{F(x)}, VK_{F(x)}, PK_F, PP)$;
10: **if** $(\hat{y} = F(x))$ **then**
11: **return** 1
12: **else**
13: **return** 0

mean to say that they cannot ascertain correctness however). The challenger chooses an input value, x, at random from the domain of F and generates an encoded input for computing $F(x)$. He also simulates a computational server S, runs $\mathsf{Compute}$ on the encoded input and gives the resulting output and verification key to the adversary who must output a guess for the value of $F(x)$. We provide $\mathcal{A}$ with oracle access as in the selective Public Verifiability game.

We must be somewhat careful about the data labels $l(\cdot)$ with regards to this notion. The label is uniquely defined for each possible input data. If this unique label enables an adversary to identify and learn the actual value of the input data then it may trivially apply F to this data and win the game. A simple method to solve this is to restrict the labelling function such that it describes the data in a non-invertible manner (e.g. the label may contain a cryptographic hash of the data). Alternatively, based on the actual CP-ABE construction being used, it may be possible to avoid this problem. For example, as in a predicate encryption scheme, the functions could hide the input attributes or policies such that it is not possible to learn the input data. In many CP-ABE schemes, including [12], the ciphertext contains a description of the policy whilst the decryption key does not immediately reveal the input attributes. In this game, we choose not to provide the adversary with the encoded inputs (ciphertexts) or the evaluation key to avoid giving the descriptive label that could be inverted to learn the input data. We will explore these restrictions further in future work.

Definition 4. *The* advantage *of a PPT adversary $\mathcal{A}$ making a polynomial number of queries q in the Blind Verification Experiment is defined as:*

$$Adv_{\mathcal{A}}^{BVerif}(\mathcal{VDC}, F, 1^{\kappa}, q) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{BVerif}[\mathcal{VDC}, F, 1^{\kappa}] = 1] - \max_{y \in \mathsf{Ran}(F)} (\Pr_{x \in \mathsf{Dom}(F)}[F(x) = y]).$$

A VDC is secure against Blind Verification *for a function F, if for all PPT adversaries $\mathcal{A}$, $Adv_{\mathcal{A}}^{BVerif}(\mathcal{VDC}, F, 1^{\kappa}, q) \leq \mathrm{negl}(\kappa)$.*

Clearly, the adversary can trivially make a guess for $F(x)$ based on a priori knowledge of the distribution of F over all possible inputs. Unless F is balanced (i.e. outputs 1 exactly half the time), the adversary could gain an advantage. Thus, we define security by subtracting the most likely guess for $F(x)$.

2.3 Instantiation

Informally, the delegator will choose a random message from the message space $\mathcal{M}$ to act as a verification token and encrypt this using a CP-ABE scheme under the Boolean[3] function F to

[3]Following Parno et al. [29] we restrict our attention to Boolean functions, and in particular the complexity class NC^1 which includes all circuits of depth $\mathsf{O}(\log \mathsf{n})$ including many common operations. Note, n-bit outputs can be implemented by n Boolean functions that use a mask to produce each bit of output in turn.

be evaluated. Each server is given a decryption key for the data x that they hold. The server attempts to decrypt the ciphertext and learns the chosen message if and only if $F(x) = 1$. By the security of the CP-ABE scheme, servers learn nothing about the message if $F(x) = 0$ since this corresponds to an access structure not being satisfied. Thus, if the correct message is returned, the delegator is convinced that $F(x) = 1$.

If, however, $F(x) = 0$, the decryption will return $\perp$. This is insufficient for verification since any server can return $\perp$ to convince a delegator of a false negative result. Thus, we produce two CP-ABE ciphertexts. As above, one corresponds to F, whilst the other corresponds to $\overline{F} = F(x) \oplus 1$ (which always outputs the opposite result to F for Boolean functions). Thus, if $F(x) = 0$ then, necessarily, $\overline{F}(x) = 1$.Hence, the server's key for data x will decrypt *exactly one* ciphertext and the returned message will distinguish whether F or $\overline{F}$ was satisfied, and therefore the value of $F(x)$. A well formed response, (d_0, d_1), from a server, therefore, satisfies the following:

$$(d_0, d_1) = \begin{cases} (m_0, \perp), & \text{if } F(x) = 1; \\ (\perp, m_1), & \text{if } F(x) = 0. \end{cases} \tag{1}$$

If the returned plaintext does not match the chosen random message then the server has returned an incorrect result (also if both results are $\perp$ but a rational malicious server would never return this)Public Verifiability is achieved by publishing a token comprising a one-way function g applied to both plaintexts. Any entity can apply g to the server's response and compare with this token to check correctness. For blind verification, a random bit b permutes the ciphertexts thus hiding whether the matched plaintext is associated with F or $\overline{F}$. The public parameters contains a two-dimensional array L_{Reg} where the first dimension, $L_{Reg}[S_i][0]$ is indexed by server identities and contains signature verification keys, whilst the second dimension, $L_{Reg}[S_i][1]$, lists functions and labels for which S_i is certified.

Although superficially similar to the PVC construction of Parno et al. [29], we consider adversaries that have access to multiple keys and must ensure that a key for different data cannot produce a valid looking response. We do this by labelling each data set x with a unique label $l(x)$ and define an attribute for each label. Then, for data x, the decryption key is formed over the attribute set $(x \cup l(x))$. During ProbGen for $F(x)$, the encryption uses the access structure encoding of the conjunction $(F \wedge l(x))$. Thus, decryption only succeeds if $F(x) = 1$ *and* the label $l(x)$ is matched in the key and ciphertext – a key for different data will not include the correct label. The instantiation of VDC is also somewhat more efficient than that for PVC since we do not require the setup of two independent ABE systems or two (expensive) key generations.

To encode an n-bit binary input string $\vec{x} = \vec{x}_1\vec{x}_2 \ldots \vec{x}_n$ as an attribute set x, we define a universe $\mathcal{U}_x = \{\mathtt{x_1}, \mathtt{x_2}, \ldots, \mathtt{x_n}\}$ of n attributes and let $\mathtt{x_i} \in x$ if and only if the i^{th} bit of the input string is 1 – that is, $x = \{\mathtt{x_i} : \vec{x}_i = 1\}$. Let $\mathcal{U} = \mathcal{U}_x \cup \mathcal{U}_l \cup \mathcal{U}_{ID}$ where $\mathcal{U}_l$ is a disjoint (from $\mathcal{U}_x$) universe representing unique labels, $l(x)$, for each input data x, and $\mathcal{U}_{ID}$ comprises server identities.[4]If such a universe becomes too large it is, of course, possible to use a *large universe* CP-ABE scheme where attributes need not be defined ahead of time. However, it is very likely that, for efficiency, the inputs and number of servers will be polynomially sized in the security parameter and can therefore be accommodated by a *small universe* construction if required.

Let $\mathcal{CPABE}$ = (ABE.Setup, ABE.KeyGen, ABE.Encrypt and ABE.Decrypt) define a CP-ABE encryption scheme over the universe $\mathcal{U}$ for a class of Boolean functions $\mathcal{F}$ closed under complement. We also make use of a signature scheme with algorithms Sig.KeyGen, Sig.Sign and Sig.Verify, and a oneway function g. Then Algorithms 1–8 define a VDC scheme for the class

[4]We assume that the following algorithms check, where relevant, that all functions and input data are formed over $\mathcal{U}_x$ and each additionally contains exactly one attribute/clause over the label universe $\mathcal{U}_l$.

of functions $\mathcal{F}$.

Alg. 1 $(PP, MK) \leftarrow \mathsf{VDC.Setup}(1^\kappa)$

1: $(MPK_{\mathrm{ABE}}, MSK_{\mathrm{ABE}}) \leftarrow \mathsf{ABE.Setup}(1^\kappa, \mathcal{U})$
2: **for** $S_i \in \mathcal{U}_{\mathrm{ID}}$ **do**
3: $\quad L_{\mathrm{Reg}}[S_i][0] = \epsilon$, $L_{\mathrm{Reg}}[S_i][1] = \{\epsilon\}$
4: $PP = (MPK_{\mathrm{ABE}}, L_{\mathrm{Reg}})$, $MK = (MSK_{\mathrm{ABE}})$

Alg. 2 $PK_F \leftarrow \mathsf{VDC.FnInit}(F, MK, PP)$

1: Set $PK_F = PP$

Alg. 3 $SK_{S_i} \leftarrow \mathsf{VDC.Register}(S_i, MK, PP)$

1: $(SK_{\mathrm{Sig}}, VK_{\mathrm{Sig}}) \leftarrow \mathsf{Sig.KeyGen}(1^\kappa)$
2: $SK_{S_i} = SK_{\mathrm{Sig}}$
3: $L_{\mathrm{Reg}}[S_i][0] = L_{\mathrm{Reg}}[S_i][0] \cup VK_{\mathrm{Sig}}$

Alg. 4 $EK_{x_i,S_i} \leftarrow \mathsf{VDC.Certify}(S_i, x_i, l(x_i), \mathcal{F}_i, MK, PP)$

1: **for** $F_j \in \mathcal{F}_i$ **do**
2: $\quad L_{\mathrm{Reg}}[S_i][1] = L_{\mathrm{Reg}}[S_i][1] \cup (F_j, l(x_i))$
3: $SK_{\mathrm{ABE},x_i} \leftarrow \mathsf{ABE.KeyGen}((x_i \cup l(x_i)), MSK_{\mathrm{ABE}}, MPK_{\mathrm{ABE}})$
4: $EK_{x_i,S_i} = SK_{\mathrm{ABE},x_i}$

Alg. 5 $(\sigma_{F(x_i)}, VK_{F(x_i)}, RK_{F(x_i)}) \leftarrow \mathsf{VDC.ProbGen}(F, l(x_i), PK_F, PP)$

1: $(m_0, m_1) \xleftarrow{\$} \mathcal{M} \times \mathcal{M}$, $b \xleftarrow{\$} \{0,1\}$
2: $c_b \leftarrow \mathsf{ABE.Encrypt}((F \wedge l(x_i)), m_b, MPK_{\mathrm{ABE}})$
3: $c_{1-b} \leftarrow \mathsf{ABE.Encrypt}((\overline{F} \wedge l(x_i)), m_{1-b}, MPK_{\mathrm{ABE}})$
4: **return** $\sigma_{F(x_i)} = (c_b, c_{1-b})$, $VK_{F(x_i)} = (g(m_b), g(m_{1-b}), L_{Reg})$ and $RK_{F(x_i)} = b$

Alg. 6 $\theta_{F(x_i)} \leftarrow \mathsf{VDC.Compute}(\sigma_{F(x_i)}, EK_{x_i,S_i}, SK_{S_i}, PP)$

1: Parse $EK_{x_i,S_i} = SK_{\mathrm{ABE},x_i}$ and $\sigma_{F(x_i)} = (c_b, c_{1-b})$
2: $d_b \leftarrow \mathsf{ABE.Decrypt}(c_b, SK_{\mathrm{ABE},x_i}, MPK_{\mathrm{ABE}})$
3: $d_{1-b} \leftarrow \mathsf{ABE.Decrypt}(c_{1-b}, SK_{\mathrm{ABE},x_i}, MPK_{\mathrm{ABE}})$
4: $\gamma \leftarrow \mathsf{Sig.Sign}((d_b, d_{1-b}, S_i), SK_{S_i})$
5: **return** $\theta_{F(x_i)} = (d_b, d_{1-b}, S_i, \gamma)$

Alg. 7 $(RT_{F(x_i)}$ or $\perp) \leftarrow \mathsf{VDC.BVerif}(\theta_{F(x_i)}, VK_{F(x_i)}, PP)$

1: Parse $VK_{F(x_i)} = (VK, VK', L_{Reg})$ and $\theta_{F(x_i)} = (d, d', S_i, \gamma)$
2: **if** $((F_j, l(x_i)) \in L_{\mathrm{Reg}}[S_i][1])$ and $(\mathsf{Sig.Verify}((d, d', S_i), \gamma, L_{\mathrm{Reg}}[S][0]) \rightarrow \mathsf{accept})$ **then**
3: $\quad$ **if** $VK = g(d)$ **then return** $RT_{F(x_i)} = d$
4: $\quad$ **if** $VK' = g(d')$ **then return** $RT_{F(x_i)} = d'$
5: **return** $RT_{F(x_i)} = \perp$

Alg. 8 $y_{F(x_i)} \leftarrow \mathsf{VDC.Retrieve}(RT_{F(x_i)}, VK_{F(x_i)}, RK_{F(x_i)}, PP)$

1: Parse $VK_{F(x_i)} = (g(m_b), g(m_{1-b}), L_{\mathrm{Reg}})$, $RT_{F(x_i)}$ and $RK_{F(x_i)} = b$
2: **if** $g(RT_{F(x_i)}) = g(m_0)$ **then return** $y_{F(x_i)} = 1$
3: **if** $g(RT_{F(x_i)}) = g(m_1)$ **then return** $y_{F(x_i)} = 0$
4: **return** $y_{F(x_i)} = \perp$

Theorem 1. *Given a secure* sIND-CPA *CP-ABE scheme for a class of Boolean functions $\mathcal{F}$ closed under complement, a one-way function g, and a signature scheme secure against* EUF-CMA*, let $\mathcal{VDC}$ be the verifiable delegable computation scheme defined in Algorithms 1–8. Then $\mathcal{VDC}$ is secure in the sense of selective Public Verifiability (Game 2) and Blind Verification (Game 3).*

Informally, Public Verifiability relies on the IND-CPA property of the CP-ABE encryption and the one-wayness of g. The proof proceeds by showing that, for the unsatisfied function F or $\overline{F}$, an adversary cannot observe if the plaintext is altered. Thus, the verification key can be the one-way function challenge $g(w)$ and the plaintext can be implicitly set to be w. A successful adversary returns w to break the one-wayness of g. Blind Verification relies on a standard probability argument.

2.3.1 Proof of Public Verifiability

Lemma 1. *$\mathcal{VDC}$ as defined by Algorithms 1–8 is secure against selective Public Verifiability (Game 2) under the same assumptions as in Theorem 1.*

Proof. Suppose $\mathcal{A}_{VDC}$ is an adversary with non-negligible advantage against the selective Public Verifiability game (Game 2) when instantiated with Algorithms 1–8. We begin by defining the following three games:

- **Game 0.** This is the selective Public Verifiability game as defined in Game 2.
- **Game 1.** This is the same as **Game 0** with the modification that in $\mathsf{ProbGen}$, we no longer return an encryption of m_0 and m_1. Instead, we choose another random message $m' \neq m_0, m_1$ and, if $F(x_i^\star) = 1$, we replace c_1 by $\mathsf{ABE.Encrypt}((\overline{F} \wedge l(x_i^\star)), m', MPK_{\text{ABE}})$. Otherwise, we replace c_0 by $\mathsf{ABE.Encrypt}((F \wedge l(x_i^\star)), m', MPK_{\text{ABE}})$. In other words, we replace the ciphertext associated with the unsatisfied function with the encryption of a separate random message unrelated to the other system parameters, and in particular to the verification keys.
- **Game 2.** This is the same as **Game 1** with the exception that instead of choosing a random message m', we implicitly set m' to be the challenge input w in the one-way function game.

We show that an adversary with non-negligible advantage against the selective Public Verifiability game can be used to construct an adversary that may invert the one-way function g.

Game 0 to Game 1. We begin by showing that there is a negligible distinguishing advantage between **Game 0** and **Game 1**. Suppose otherwise, that $\mathcal{A}_{VC}$ can distinguish the two games with non-negligible advantage δ. We then construct an adversary $\mathcal{A}_{ABE}$ that uses $\mathcal{A}_{VDC}$ as a sub-routine to break the selective IND-CPA security of the CP-ABE scheme. We consider a challenger $\mathcal{C}$ playing the IND-CPA game with $\mathcal{A}_{ABE}$, who in turn acts as a challenger in the Verifiability game for $\mathcal{A}_{VDC}$:

1. $\mathcal{A}_{VDC}$ is given the security parameter by the environment, and declares (to $\mathcal{A}_{ABE}$) its choice of input data $x_i^\star$ and data label $l(x_i^\star)$.
2. $\mathcal{A}_{ABE}$ must send a challenge access structure to the challenger. It first computes $r = F(x_i^\star)$ – that is, the outcome of the challenge function F applied to the challenge input data. If $r = 1$, $\mathcal{A}_{ABE}$ sets $\mathbb{A}^\star = (\overline{F} \wedge l(x_i^\star))$. Else, $r = 0$, $\mathbb{A}^\star = (F \wedge l(x_i^\star))$.

3. $\mathcal{C}$ runs the $\mathsf{ABE.Setup}$ algorithm on the security parameter to generate MPK_{ABE} and MSK_{ABE}. He gives MPK_{ABE} to $\mathcal{A}_{ABE}$.

4. $\mathcal{A}_{ABE}$ now simulates running $\mathsf{VDC.Setup}$ such that the outcome is consistent with MPK_{ABE}. It initializes the list L_{Reg} and sets $PP = (MPK_{\text{ABE}}, L_{Reg})$. The master key is implicitly set to MSK_{ABE}. $\mathcal{A}_{ABE}$ also runs FnInit as given in the construction.

5. To generate the challenge input, $\mathcal{A}_{ABE}$ begins by choosing a random bit b, three random messages m_0, m_1 and m' from the message space, and another random bit t.

   $\mathcal{A}_{ABE}$ sends the messages m_0 and m_1 to $\mathcal{C}$ as the challenge messages for the CP-ABE game. $\mathcal{C}$ chooses a random bit c and returns $CT^\star \leftarrow \mathsf{Encrypt}(m_c, \mathbb{A}^\star, MPK_{\text{ABE}})$.

   - If $r = 1$ (that is, $\mathbb{A}^\star = (\overline{F} \wedge l(x_i^\star))$), $\mathcal{A}_{ABE}$ generates $c_b \leftarrow \mathsf{Encrypt}((F \wedge l(x_i^\star)), m', MPK_{\text{ABE}})$ and sets $c_{1-b} = CT^\star$ (formed over $\mathbb{A}^\star$ by $\mathcal{C}$). It also sets $VK_b = g(m')$ and $VK_{1-b} = g(m_t)$.
   - Else $r = 0$, and $\mathcal{A}_{ABE}$ sets $c_b = CT^\star$ and computes $c_{1-b} \leftarrow \mathsf{Encrypt}((\overline{F} \wedge l(x_i^\star)), m', MPK_{\text{ABE}})$. It sets $VK_b = g(m_t)$ and $VK_{1-b} = g(m')$.

   Finally, $\mathcal{A}_{ABE}$ sets $\sigma_{F(x_i^\star)} = (c_b, c_{1-b})$, $VK_{F(x)} = (VK_b, VK_{1-b})$ and $RK_{F(x_i^\star)} = b$.

6. $\mathcal{A}_{ABE}$ sends the output from $\mathsf{ProbGen}$ along with the public information to $\mathcal{A}_{VC}$, who is also given oracle access to which $\mathcal{A}_{ABE}$ responds as follows:

   - $\mathsf{FnInit}(\cdot, MK, PP)$: run as per Algorithm 2.
   - $\mathsf{Register}(\cdot, MK, PP)$: run as per Algorithm 3.
   - $\mathsf{Certify}(\cdot, \cdot, \cdot, \cdot, MK, PP)$: To generate the evaluation key for the queried attribute set x, $\mathcal{A}_{ABE}$ makes use of the KeyGen oracle in the CP-ABE game. It first updates L_{Reg} as in lines 1–2 of the $\mathsf{Certify}$ algorithm. Then it sets $x' = x \cup l(x)$ and makes an oracle query to $\mathcal{C}$ for $\mathcal{O}^{\mathsf{KeyGen}}(x', MK, PK)$ as in Oracle Query 5. $\mathcal{C}$ shall generate a CP-ABE decryption key $SK_{x'}$ for x if and only if $x' \notin \mathbb{A}^\star$. Now, since each data label is unique, $l(x) = l(y)$ if and only if $x = y$. By the definition of $\mathbb{A}^\star$, x' will satisfy $\mathbb{A}^\star$ only if the data labels $l(x)$ and $l(x_i^\star)$ match, hence only if $x = x_i^\star$. Now, if $x = x_i^\star$, then additionally, x must satisfy either F or $\overline{F}$ as chosen in $\mathbb{A}^\star$ in Step 2. However, this was chosen specifically such that $x_i^\star$ (and therefore x) does not satisfy the function, and therefore $x' \notin \mathbb{A}^\star$ and $\mathcal{C}$ may generate the key, which $\mathcal{A}_{ABE}$ will receive as EK_{x,S_i}.

7. Eventually, $\mathcal{A}_{VC}$ outputs $\theta_{F(x_i^\star)}$ which it believes is a valid forgery (i.e. that it will be accepted yet does not correspond to the correct value of $F(x_i^\star)$).

8. $\mathcal{A}_{ABE}$ parses $\theta_{F(x_i^\star)}$ as $(d_b, d_{1-b}, S_i, \gamma)$ and using the retrieval key $RK_{F(x_i^\star)} = b$, finds d_0 and d_1. One of d_0 and d_1 will be $\perp$ (by construction) and we denote the other value by Y.

   Observe that, since $\mathcal{A}_{VC}$ is assumed to be a successful adversary against selective public verifiability, the non-$\perp$ value, Y, that it will return will be the plaintext m_c since the challenge access structure was always set to be unsatisfied on the challenge input.

   Thus, if $g(Y) = g(m_t)$, $\mathcal{A}_{ABE}$ outputs a guess $c' = t$ and otherwise guesses $c' = (1 - t)$.

If $t = c$ (the challenge bit chosen by $\mathcal{C}$), we observe that the above corresponds to **Game 0** (since the verification key comprises $g(m')$ where m' is the message a legitimate server could

recover, and $g(m_c)$ where m_c is the other plaintext). Alternatively, $t = 1 - c$ and the distribution of the above experiment is identical to **Game 1** (since the verification key comprises the legitimate message and a random message m_{1-c} that is unrelated to the ciphertext).

Now, we consider the advantage of this constructed $\mathcal{A}_{ABE}$ playing the sIND-CPA game for CP-ABE: Recall that by assumption, $\mathcal{A}_{VC}$ has a non-negligible advantage δ in distinguishing between **Game 0** and **Game 1** – that is

$$|\Pr(\mathbf{Exp}^0_{\mathcal{A}_{VC}}[\mathcal{VDC}, F, 1^\kappa]) - \Pr(\mathbf{Exp}^1_{\mathcal{A}_{VC}}[\mathcal{VDC}, F, 1^\kappa])| \geqslant \delta$$

where $\mathbf{Exp}^i_{\mathcal{A}_{VC}}[\mathcal{VDC}, F, 1^\kappa]$ denotes the output of running $\mathcal{A}_{VC}$ in **Game** i.

$$\begin{aligned}
\Pr(c' = c) &= \Pr(t = c)\Pr(c' = c | t = c) + \Pr(t \neq c)\Pr(c' = c | t \neq c) \\
&= \frac{1}{2}\Pr(g(Y) = g(m_t) | t = c) + \frac{1}{2}\Pr(g(Y) \neq g(m_t) | t \neq c) \\
&= \frac{1}{2}\Pr(\mathbf{Exp}^0_{\mathcal{A}_{VC}}[\mathcal{VDC}, F, 1^\kappa]) + \frac{1}{2}\left(1 - \Pr(g(\sigma_{y^\star}) = g(m_t) | t \neq c)\right) \\
&= \frac{1}{2}\Pr(\mathbf{Exp}^0_{\mathcal{A}_{VC}}[\mathcal{VDC}, F, 1^\kappa]) + \frac{1}{2}\left(1 - \Pr(\mathbf{Exp}^1_{\mathcal{A}_{VC}}[\mathcal{VDC}, F, 1^\kappa])\right) \\
&= \frac{1}{2}\left(\Pr(\mathbf{Exp}^0_{\mathcal{A}_{VC}}[\mathcal{VDC}, F, 1^\kappa]) - \Pr(\mathbf{Exp}^1_{\mathcal{A}_{VC}}[\mathcal{VDC}, F, 1^\kappa]) + 1\right) \\
&\geqslant \frac{1}{2}(\delta + 1)
\end{aligned}$$

Hence,

$$\begin{aligned}
Adv_{\mathcal{A}_{ABE}} &\geqslant \left|\Pr(c = c') - \frac{1}{2}\right| \\
&\geqslant \left|\frac{1}{2}(\delta + 1) - \frac{1}{2}\right| \\
&\geqslant \frac{\delta}{2}
\end{aligned}$$

Hence, if $\mathcal{A}_{VC}$ has advantage δ at distinguishing these games then $\mathcal{A}_{ABE}$ can win the sIND-CPA game for CP-ABE with non-negligible probability. Thus since we assumed the CP-ABE scheme to be secure, we conclude that $\mathcal{A}_{VC}$ cannot distinguish **Game 0** from **Game 1** with non-negligible probability.

Game 1 to Game 2. The transition from **Game 1** to **Game 2** is simply to set the value of m' to no longer be random but instead to correspond to the challenge w in the one-way function inversion game. We argue that the adversary has no distinguishing advantage between these games since the new value is independent of anything else in the system bar the verification key $g(w)$ and hence looks random to an adversary with no additional information (in particular, $\mathcal{A}_{VC}$ does not see the challenge for the one-way function as this is played between $\mathcal{C}$ and $\mathcal{A}_{ABE}$).

Final Proof. We now show that using $\mathcal{A}_{VC}$ in **Game 2**, $\mathcal{A}_{ABE}$ can invert the one-way function g – that is, given a challenge $z = g(w)$ we can recover w. Specifically, during ProbGen, we choose the messages as follows:

- if $F(x) = 1$, we implicitly set m_{1-b} to be w and set the verification key component $VK_{1-b} = z$. We choose m_b and VK_b randomly as usual.

- if $F(x) = 0$, we implicitly set m_b to be w and set the verification key component $VK_b = z$. We choose m_{1-b} and VK_{1-b} randomly as usual.

Now, since $\mathcal{A}_{VC}$ is assumed to be successful, it will output a forgery comprising the plaintext encrypted under the unsatisfied function (F or $\overline{F}$). By construction, this will be w (and the adversary's view is consistent since the verification key is simulated correctly using z). $\mathcal{A}_{ABE}$ can therefore forward this result to $\mathcal{C}$ in order to invert the one-way function with the same non-negligible probability that $\mathcal{A}_{VC}$ has against the public verifiability game.

We conclude that if the ABE scheme is sIND-CPA secure and the one-way function is hard-to-invert, then $\mathcal{VDC}$ as defined by Algorithms 1–8 is secure in the sense of selective Public Verifiability.

□

2.3.2 Blind Verification

Lemma 2. *$\mathcal{VDC}$ as defined by Algorithms 1–8 is secure against Blind Verification (Game 3) under the same assumptions as in Theorem 1.*

Proof. We show that if the underlying CP-ABE scheme is sIND-CPA secure then the construction in Algorithms 1–8 is secure against Blind Verification. The proof follows from a standard probability argument. We first argue that only $\theta_{F(x)}$ and $VK_{F(x)}$ may reveal useful information to the adversary. We then show that the adversarial view of these inputs does not provide an advantage at guessing the result.

Over the course of the game, the adversary sees the following inputs: $\theta_{F(x)}$, $VK_{F(x)}$, PK_F, PP and the outputs from oracle queries. By construction, $PK_F = PP$ and as the public parameters are defined at the beginning of the game, this clearly does not reveal any information about $F(x)$. In particular, since the adversary does not see the encoded input (ciphertexts) from the challenge computation, the ABE public parameters in PP are not helpful (else the ABE scheme would not be IND-CPA secure), and neither is the list L_{Reg} which contains only function lists and signature verification keys.

The inputs $\theta_{F(x)}$ and $VK_{F(x)}$ clearly do rely on the values of x and $F(x)$ and we will consider these shortly. We first consider the oracle access given to the following functions:

- $\mathsf{FnInit}(\cdot, MK, PP)$: FnInit queries simply return the public parameters which we considered previously as an explicit adversarial input.
- $\mathsf{Register}(\cdot, MK, PP)$: Queries to this oracle generate a signing key for a server S_i. However, this does not relate to the retrieval key or the choice of x.
- $\mathsf{Certify}(\cdot, \cdot, \cdot, \cdot, MK, PP)$: A call to this oracle will add an entry to L_{Reg} comprising a function identifier and a data label (which we have assumed not to leak the input value itself). It also creates a decryption key for the underlying ABE system. Again, as the adversary only sees plaintexts and does not see the ciphertexts forming the challenge encoded input, such a key is not useful.

Hence, oracle access to these functions does not help the adversary to distinguish which input was selected and hence the value of $F(x)$. Thus, the only inputs to the adversary that depend on the choice of challenge input are $\theta_{F(x)}$ and $VK_{F(x)}$, and so we restrict our attention to these.

Recall that a well-formed response by the server will be either $(m_b, \perp)$ or $(\perp, m_{1-b})$ according to $RK_{F(x)}$. In detail this means, where $RK_{F(x)} = b$:

- if $F(x) = 1$, $\theta_{F(x)} = \begin{cases} (m_0, \perp), & \text{if } b = 0 \\ (\perp, m_0), & \text{if } b = 1 \end{cases}$

- if $F(x) = 0$, $\theta_{F(x)} = \begin{cases} (\perp, m_1), \text{ if } b = 0 \\ (m_1, \perp), \text{ if } b = 1 \end{cases}$

Note that $VK_{F(x)} = (g(m_b), g(m_{1-b}))$ by definition (excluding L_{Reg} which we discussed above). We denote by $\mathcal{V}$ the adversary's view of $\theta_{F(x)}$ and $VK_{F(x)}$ – that is, $\mathcal{V} = (d_b, d_{1-b}, g(m_b), g(m_{1-b}))$ if $\theta_{F(x)} = (d_b, d_{1-b})$ and $VK_{F(x)} = (g(m_b), g(m_{1-b}))$.

We now show that the probability of the adversary correctly guessing the value of $F(x)$ given a particular view $\mathcal{V}$ is identical to his success at guessing without seeing $\mathcal{V}$. Thus, he has no advantage at guessing $F(x)$ over his a priori knowledge of the distribution of F.

Let $\mathcal{V}_1 = (m', \perp, g(m'), g(m_{1-b}))$ and let $\mathcal{V}_2 = (\perp, m'', g(m_b), g(m''))$. We claim that these are the only possible views – $\mathcal{A}$ sees one message (either m_0 or m_1, both drawn uniformly from the same distribution) along with g applied to that message and to a different (unseen) message.

We begin by noting the following facts: (i) the value of $F(x)$ and of $b \xleftarrow{\$} \{0,1\}$ are independent events; (ii) $\Pr[b = 1] = \frac{1}{2}$; and (iii) $\Pr[F(x) = 0] + \Pr[F(x) = 1] = 1$ since F is a Boolean function. Now,

$$\begin{aligned}
\Pr[\mathcal{V} = \mathcal{V}_1] &= \Pr[(F(x) = 1 \wedge b = 0) \vee (F(x) = 0 \wedge b = 1)] \\
&= \Pr[F(x) = 1 \wedge b = 0] + \Pr[F(x) = 0 \wedge b = 1] \\
&= \Pr[F(x) = 1] \Pr[b = 0] + \Pr[F(x) = 0] \Pr[b = 1] \text{ by (i)} \\
&= \frac{1}{2} \Pr[F(x) = 1] + \frac{1}{2} \Pr[F(x) = 0] \\
&= \frac{1}{2} (\Pr[F(x) = 0] + \Pr[F(x) = 1]) \\
&= \frac{1}{2}
\end{aligned} \tag{2}$$

Now,

$$\begin{aligned}
\Pr[F(x) = 0 | \mathcal{V} = \mathcal{V}_1] &= \frac{\Pr[F(x) = 0 \wedge \mathcal{V} = \mathcal{V}_1]}{\Pr[\mathcal{V} = \mathcal{V}_1]} \\
&= \frac{\Pr[F(x) = 0 \wedge b = 1]}{\Pr[\mathcal{V} = \mathcal{V}_1]} \\
&= \frac{\Pr[F(x) = 0] \Pr[b = 1]}{\Pr[\mathcal{V} = \mathcal{V}_1]} \text{ by (i)} \\
&= \frac{\frac{1}{2} \Pr[F(x) = 0]}{\frac{1}{2}} \text{ by (2)} \\
&= \Pr[F(x) = 0]
\end{aligned}$$

Similarly,

$$\begin{aligned}
\Pr[\mathcal{V} = \mathcal{V}_2] &= \Pr[(F(x) = 1 \wedge b = 1) \vee (F(x) = 0 \wedge b = 0)] \\
&= \Pr[F(x) = 1 \wedge b = 1] + \Pr[F(x) = 0 \wedge b = 0] \\
&= \Pr[F(x) = 1] \Pr[b = 1] + \Pr[F(x) = 0] \Pr[b = 0] \text{ by (i)} \\
&= \frac{1}{2} \Pr[F(x) = 1] + \frac{1}{2} \Pr[F(x) = 0] \\
&= \frac{1}{2} (\Pr[F(x) = 0] + \Pr[F(x) = 1]) \\
&= \frac{1}{2}
\end{aligned} \tag{3}$$

Now,

$$\begin{aligned}
\Pr[F(x) = 0 | \mathcal{V} = \mathcal{V}_2] &= \frac{\Pr[F(x) = 0 \wedge \mathcal{V} = \mathcal{V}_2]}{\Pr[\mathcal{V} = \mathcal{V}_2]} \\
&= \frac{\Pr[F(x) = 0 \wedge b = 0]}{\Pr[\mathcal{V} = \mathcal{V}_2]} \\
&= \frac{\Pr[F(x) = 0] \Pr[b = 0]}{\Pr[\mathcal{V} = \mathcal{V}_2]} \text{ by (i)} \\
&= \frac{\frac{1}{2} \Pr[F(x) = 0]}{\frac{1}{2}} \text{ by (3)} \\
&= \Pr[F(x) = 0]
\end{aligned}$$

A symmetric argument holds for $F(x) = 1$, and hence we conclude that knowledge of the adversarial inputs provides no advantage in determining $F(x)$ other than that which could be guessed without that knowledge (i.e. the inputs leak no information about $F(x)$). □

3 Hybrid Publicly Verifiable Computation

We have seen how CP-ABE can be applied in a similar fashion to PVC schemes using KP-ABE [29, 2] to construct VDC. We now unify these notions under the umbrella of *Hybrid Publicly Verifiable Compuation* (HPVC). This is a single system (with associated costs of a single setup operation and system parameters) that can flexibly handle multiple modes of operation. Thus, within an organization, a single KDC may initialize an HPVC system that provides functionality for many users with diverse requirements. We give formal definitions and a provably secure construction based on a novel use of DP-ABE. The key observation is that DP-ABE can, using special attribute tokens, implement KP-ABE, CP-ABE or DP-ABE policies. In more detail, we capture the following functionality:

- **RPVC:** Uses KP-ABE policies only to achieve RPVC (Sect. 1.1). Thus weak delegators may outsource computations and receive verifiably correct results;
- **RPVC with Access Control:** Uses DP-ABE policies to achieve RPVC with restrictions on the servers that may perform a computation. This will be discussed further in Sect. 4;
- **VDC:** Uses CP-ABE policies only to achieve the verifiable delegation of a function to servers owning their own input data as discussed in Sect. 2.

3.1 Revocable Dual-policy Attribute-based Encryption

Dual-Policy Attribute-Based Encryption (DP-ABE) [7] conjunctively combines Key-Policy ABE (KP-ABE) [25] and Ciphertext-Policy ABE (CP-ABE) [12]. Both the ciphertext and the decryption key comprise an attribute set *and* an access policy. Thus, the ciphertext is associated with both an subjective access policy (as per CP-ABE) detailing which entities may decrypt *and* an objective attribute set describing the data. Decryption keys comprise an objective access policy (as per KP-ABE) and a subjective attribute set. Decryption succeeds when *both* attribute sets satisfy their corresponding access policies.

For our HPVC instantiation, we require the new notion of revocable DP-ABE scheme which we introduce by combining the DP-ABE construction [7] with indirectly revocable KP-ABE [5]. Full details can be found can be found in App. B. We observe that it is sufficient to revoke the policy in *either* the key *or* the ciphertext – decryption succeeds if and only if *both* attribute sets

satisfy their corresponding access structure. To prevent decryption, at least one attribute set should not satisfy the corresponding access structure. This may be viewed as a consequence of De Morgan's law: decryption succeeds if both $(\omega \in \mathbb{O}) \wedge (\psi \in \mathbb{S})$. Hence to prevent decryption, we require $\neg((\omega \in \mathbb{O}) \wedge (\psi \in \mathbb{S})) = (\omega \notin \mathbb{O}) \vee (\psi \notin \mathbb{S})$. Here we define revocable DP-ABE using indirect revocation in the key-policy (maintaining consistency with [2]). Revocable KP-ABE and CP-ABE can be found by ignoring relevant parameters.

Definition 5 (Revocable Key DP-ABE)**.** *A* Dual-Policy Attribute-based Encryption scheme with revocation in the key-policy *comprises five algorithms:*

- $(PP, MK) \leftarrow \mathsf{Setup}(1^\kappa)$*: Takes the security parameter as input and generates public parameters PP for the system and a master secret key MK.*
- $CT_{(\omega,\mathbb{S}),t} \leftarrow \mathsf{Encrypt}(m, (\omega, \mathbb{S}), t, PP)$*: Takes in the public parameters, a message to be encrypted, a subjective access policy $\mathbb{S}$ and an objective attribute set ω. It outputs a ciphertext that is valid for time t.*
- $\mathrm{SK}_{(\mathbb{O},\psi),\mathrm{ID}} \leftarrow \mathsf{KeyGen}(\mathrm{ID}, (\mathbb{O}, \psi), \mathrm{MK}, \mathrm{PP})$*: Takes the public parameters and master secret key, an identity* ID *as well as an objective access policy $\mathbb{O}$ and a subjective attribute set ψ. It outputs a secret decryption key* $\mathrm{SK}_{(\mathbb{O},\psi),\mathrm{ID}}$*.*
- $UK_{R,t} \leftarrow \mathsf{KeyUpdate}(R, t, MK, PP)$*: Takes a revocation list R that contains the identities of revoked entities, the current time period, as well as the public parameters and master secret key. It outputs updated key material $UK_{R,t}$.*
- m *or* $\perp \leftarrow \mathsf{Decrypt}(CT_{(\omega,\mathbb{S}),t}, (\omega, \mathbb{S}), (\mathbb{O}, \psi), \mathrm{SK}_{(\mathbb{O},\psi),\mathrm{ID}}, UK_{R,t}, PP)$*: Takes as input the public parameters, a ciphertext and key with associated policies and attribute sets, and updated key material. It outputs the correct plaintext m if and only if the f objective attributes ω satisfies the objective access structure* $\mathbb{O}$ and *the subjective attributes ψ satisfies the subjective policy* $\mathbb{S}$ and *the value of t in the update key matches that specified during encryption.*

The necessary background, security definition, full construction as well as a security proof can be found in Appendix B.

3.2 Specification

We give a generic definition in terms of objective (RPVC) and subjective (VDC) policies ($\mathbb{O}$ and $\mathbb{S}$) and corresponding attribute sets (ω and ψ respectively). These depend upon the mode in which the algorithm is being run, and are detailed in Table 1.[5] Informally, a HPVC scheme for a family of functions $\mathcal{F}$ begins with a trusted Key Distribution Center (KDC) (e.g. a trusted third party or a delegator) setting up the system by producing public parameters and a master secret key. For each function of interest F, the KDC provides a public delegation key PK_F and maintains a list L_{Reg} of servers willing to provide a computation service for F along with a unique label l (in VDC this describes the data held by those servers). Next, the KDC registers each server S_i by deriving a private signing key SK_{S_i}. The $\mathsf{Certify}$ algorithm allows each server S_i to provide a list of functions $\mathcal{F}_i$ that they wish to provide computational services for, and the KDC generates an evaluation key $EK_{(\mathbb{O},\psi),S_i}$ and updates L_{Reg} accordingly. Depending on the mode, we choose the values of $\mathbb{O}$ and ψ as given in Table 1. A delegator runs $\mathsf{ProbGen}$ to outsource the computation of $F(x)$; again the values of ω and $\mathbb{S}$ depend on the mode. The algorithm generates

[5]The meaning of T_O and T_S will become clear in Section 3.4 but for now it suffices to consider these as *null* terms.

Table 1: Parameter definitions for different modes

mode	$\mathbb{O}$	ψ	ω	$\mathbb{S}$	l	$\mathcal{F}_i$
RPVC	F	$\{T_S\}$	x	$\{\{T_S\}\}$	$l(F)$	$\{F\}$
VDC	$\{\{T_O\}\}$	x	$\{T_O\}$	F	$l(x)$	$\{F_1, \ldots, F_m\}$
PVC-AC	F	s	x	P	$l(F)$	$\{F\}$

an encoded input $\sigma_{(\omega,\mathbb{S})}$, a public verification key $VK_{(\omega,\mathbb{S})}$ and an output retrieval key $RK_{(\omega,\mathbb{S})}$. A server may use the encoded input with their evaluation key $EK_{(\mathbb{O},\psi),S_i}$ to compute an output $\theta_{(\mathbb{O},\psi),(\omega,\mathbb{S})}$ encoding $F(x_i)$. Verification comprises two steps. Blind Verification is performed by *any* user using the verification key $VK_{(\omega,\mathbb{S})}$ to verify correctness of the result *without* learning the output value, and generates an output retrieval token $RT_{(\mathbb{O},\psi),(\omega,\mathbb{S})}$. The algorithm also generates a token $\tau_{(\mathbb{O},\psi),(\omega,\mathbb{S})}$ indicating the correctness and the server ID. If verification failed, this token is sent to the KDC and the server is revoked from performing further evaluations and hence incurs a penalty. Otherwise, $RT_{(\mathbb{O},\psi),(\omega,\mathbb{S})}$ can be used in the $\mathsf{Retrieve}$ algorithm to reveal the final result $y_{(\mathbb{O},\psi),(\omega,\mathbb{S})} = F(x_i)$. An HPVC scheme is *correct* if verification almost certainly succeeds when all algorithms are run honestly.

We adapt the PVC framework introduced by Alderman et al. [2, 3].

Definition 6. *A* Hybrid Publicly Verifiable Computation (HPVC) scheme *comprises the following algorithms:*

1. $(PP, MK) \leftarrow \mathsf{Setup}(1^\kappa)$
2. $PK_F \leftarrow \mathsf{FnInit}(F, MK, PP)$
3. $SK_{S_i} \leftarrow \mathsf{Register}(S_i, MK, PP)$
4. $EK_{(\mathbb{O},\psi),S_i} \leftarrow \mathsf{Certify}(\mathtt{mode}, (\mathbb{O}, \psi), l, \mathcal{F}_i, S_i, MK, PP)$
5. $(\sigma_{(\omega,\mathbb{S})}, VK_{(\omega,\mathbb{S})}, RK_{(\omega,\mathbb{S})}) \leftarrow \mathsf{ProbGen}(\mathtt{mode}, (\omega, \mathbb{S}), l, PK_F, PP)$
6. $\theta_{(\mathbb{O},\psi),(\omega,\mathbb{S})} \leftarrow \mathsf{Compute}(\mathtt{mode}, \sigma_{(\omega,\mathbb{S})}, EK_{(\mathbb{O},\psi),S_i}, SK_{S_i}, PP)$
7. $y_{(\mathbb{O},\psi),(\omega,\mathbb{S})} \leftarrow \mathsf{Verify}(\theta_{(\mathbb{O},\psi),(\omega,\mathbb{S})}, RT_{(\mathbb{O},\psi),(\omega,\mathbb{S})}, \tau_{(\mathbb{O},\psi),(\omega,\mathbb{S})}, VK_{(\omega,\mathbb{S})}, RK_{(\omega,\mathbb{S})}, PP)$:
   - $(RT_{(\mathbb{O},\psi),(\omega,\mathbb{S})}, \tau_{(\mathbb{O},\psi),(\omega,\mathbb{S})}) \leftarrow \mathsf{BVerif}(\theta_{(\mathbb{O},\psi),(\omega,\mathbb{S})}, VK_{(\omega,\mathbb{S})}, PP)$
   - $y_{(\mathbb{O},\psi),(\omega,\mathbb{S})} \leftarrow \mathsf{Retrieve}(RT_{(\mathbb{O},\psi),(\omega,\mathbb{S})}, \tau_{(\mathbb{O},\psi),(\omega,\mathbb{S})}, VK_{(\omega,\mathbb{S})}, RK_{(\omega,\mathbb{S})}, PP)$
8. $\{EK_{(\mathbb{O},\psi),S'}\}$ *or* $\perp \leftarrow \mathsf{Revoke}(\tau_{(\mathbb{O},\psi),(\omega,\mathbb{S})}, (\mathbb{O}, \psi), (\omega, \mathbb{S}), MK, PP)$

Definition 7. *A Hybrid Publicly Verifiable Outsourced Computation* (MPVC) *scheme is* correct *for a family of functions $\mathcal{F}$ if for all functions $F \in \mathcal{F}$ and inputs x, where* $\mathrm{negl}(\cdot)$ *is a negligible function of its input:*

$$\begin{aligned}
&\Pr[(PP, MK) \leftarrow \mathsf{Setup}(1^\kappa), PK_F \leftarrow \mathsf{FnInit}(PP, MK, F), SK_{S_i} \leftarrow \mathsf{Register}(S_i, MK, PP),\\
&\quad EK_{(\mathbb{O},\psi),S_i} \leftarrow \mathsf{Certify}(\mathtt{mode}, (\mathbb{O}, \psi), l, \mathcal{F}_i, S_i, MK, PP),\\
&\quad (\sigma_{(\omega,\mathbb{S})}, VK_{(\omega,\mathbb{S})}, RK_{(\omega,\mathbb{S})}) \leftarrow \mathsf{ProbGen}(\mathtt{mode}, (\omega, \mathbb{S}), l, PK_F, PP),\\
&\quad y_{(\mathbb{O},\psi),(\omega,\mathbb{S})} \leftarrow \mathsf{Verify}(\mathsf{Compute}(\mathtt{mode}, \sigma_{(\omega,\mathbb{S})}, EK_{(\mathbb{O},\psi),S_i}, SK_{S_i}, PP), VK_{(\omega,\mathbb{S})}, RK_{(\omega,\mathbb{S})}, PP)]\\
&\quad = 1 - \mathrm{negl}(\kappa).
\end{aligned}$$

3.3 Security Models

In this section we discuss some security models which are of interested in HPVC. However, in the cases of Public Verifiability and Revocation, we also define weaker notions of security which we term *selective, semi-static* notions. This is due to the particular IND-sHRSS indirectly revocable key-policy attribute-based encryption scheme we use in our construction, which introduces similar restrictions. Thus, with our current primitives we cannot achieve full security for these notions, but can achieve the slightly weaker variants presented here. In this section we will discuss the restrictions we must impose and how they could be removed in the future.

These variants require two additional restrictions on the adversary. Firstly, the adversary must declare upfront (before seeing the public parameters) the set of input values to be used in the challenge stage. This is in contrast to the full game where the inputs are chosen after the adversary has oracle access to the system. Secondly, the adversary must (e.g. on line 5 in Game 5), declare a list $\overline{R}$ of servers that must be revoked when the challenge encoded inputs are generated from ProbGen. The adversary must do this before receiving oracle access.

The selective restriction (requiring the adversary to declare its challenge input before seeing the public parameters) stems from the selectively secure DP-ABE construction we use, whilst the semi-static restriction arises from the revocation mechanism employed (requiring the adversary to declare a list of entities to be revoked at challenge time before receiving oracle access). With our current primitives, we cannot achieve 'full security', but instead achieve the slightly weaker notion.

In the selective semi-static game, we actually have the adversary select challenge inputs for both objective and subjective policies, and both types of attribute set. Thus, we cover both flavours of functionality (RPVC and VDC) as these will, in either mode, select F, the input data, and a dummy attribute and a dummy policy (for single mode operation).

To accommodate the semi-static restriction, the challenger must define two additional parameters: Q_{Rev} and t. The counter t models system time and is incremented whenever a Revoke query is made. Keys that were generated during previous time periods should no longer be considered valid unless a fresh update key is issued. In the IND-sHRSS notion for revocable DP-ABE, the adversary could query for update keys for arbitrary time periods. In our model, however, we consider an interactive protocol in which time increases linearly and hence oracle queries use the current value of t. Similarly, an adversary against IND-sHRSS could select a time period for the challenge ciphertext. In our model, we parameterize the adversary by the number of queries made to the Revoke oracle, q_t. Thus the challenge time can be inferred in terms of q_t.

Q_{Rev} is a list of currently revoked servers (this will be more important in the Revocation game). Whereas in the IND-sHRSS game for the revocable DP-ABE scheme, the revocation list could be dynamically changed per oracle query, here we require the revocation list to remain consistent between subsequent oracle queries to model realistic system evolution. Recall that the semi-static restriction required the adversary to select a list of revoked servers $\overline{R}$ for the challenge time *before* receiving oracle access. To ensure that this list is correct at the time of creating the challenge, we compare it to the list of currently revoked servers Q_{Rev} and require that $\overline{R} \subseteq Q_{Rev}$. If not, the adversary loses as he has not performed a valid sequence of queries to match his choice of $\overline{R}$.

Otherwise, the game proceeds much as in Game 2 with updated notation to account for the general policy specification. The adversary is given access to oracles for FnInit, Register, Certify and Revoke. To avoid trivial wins, we must add additional restrictions to oracle queries made to Certify and Revoke so that the adversary may not obtain an evaluation key *and* an update key for a server that should be revoked at the time of the challenge, as specified in Oracle Queries 1

Game 4 $\mathbf{Exp}_{\mathcal{A}}^{PubVerif}[\mathcal{HPVC}, \mathtt{mode}, F, 1^{\kappa}]$:

$(PP, MK) \leftarrow \mathsf{Setup}(1^{\kappa})$
$PK_F \leftarrow \mathsf{FnInit}(F, MK, PP)$
$((\mathbb{O}^{\star}, \omega^{\star}, \mathbb{S}^{\star}, \psi^{\star}), l^{\star}) \leftarrow \mathcal{A}^{\mathcal{O}}(PK_F, PP)$
$(\sigma_{(\omega^{\star},\mathbb{S}^{\star})}, VK_{(\omega^{\star},\mathbb{S}^{\star})}, RK_{(\omega^{\star},\mathbb{S}^{\star})}) \leftarrow \mathsf{ProbGen}(\mathtt{mode}, (\omega^{\star}, \mathbb{S}^{\star}), l^{\star}, PK_F, PP)$
$\theta_{(\mathbb{O}^{\star},\psi^{\star}),(\omega^{\star},\mathbb{S}^{\star})} \leftarrow \mathcal{A}^{\mathcal{O}}(\sigma_{(\omega^{\star},\mathbb{S}^{\star})}, VK_{(\omega^{\star},\mathbb{S}^{\star})}, RK_{(\omega^{\star},\mathbb{S}^{\star})})$
$(RT_{(\mathbb{O}^{\star},\psi^{\star}),(\omega^{\star},\mathbb{S}^{\star})}, \tau_{(\mathbb{O}^{\star},\psi^{\star}),(\omega^{\star},\mathbb{S}^{\star})}) \leftarrow \mathsf{BVerif}(\theta_{(\mathbb{O}^{\star},\psi^{\star}),(\omega^{\star},\mathbb{S}^{\star})}, VK_{(\omega^{\star},\mathbb{S}^{\star})}, PP)$
$y_{(\mathbb{O}^{\star},\psi^{\star}),(\omega^{\star},\mathbb{S}^{\star})} \leftarrow \mathsf{Retrieve}(RT_{(\mathbb{O}^{\star},\psi^{\star}),(\omega^{\star},\mathbb{S}^{\star})}, \tau_{(\mathbb{O}^{\star},\psi^{\star}),(\omega^{\star},\mathbb{S}^{\star})}, VK_{(\omega^{\star},\mathbb{S}^{\star})}, RK_{(\omega^{\star},\mathbb{S}^{\star})}, PP)$
if $((y_{(\mathbb{O}^{\star},\psi^{\star}),(\omega^{\star},\mathbb{S}^{\star})}, \tau_{(\mathbb{O}^{\star},\psi^{\star}),(\omega^{\star},\mathbb{S}^{\star})} \neq (\perp, (\mathsf{reject}, \mathcal{A}))))$ **and** $(y_{(\mathbb{O}^{\star},\psi^{\star}),(\omega^{\star},\mathbb{S}^{\star})} \neq F(x)))$ **then**
  return 1
else
  return 0

and 2. A query to the Certify oracle in RPVC mode, will return $\perp$ if (i) the list of functions comprises the challenge F for a server S_i that will not be revoked at the time of the challenge, or (ii) if the query is made during the challenge time period q_t and, excluding S_i as it is about to be certified, there is a server that should be revoked according to R but is not currently. Similarly, a query to the Revoke oracle will increment the time parameter t, and return $\perp$ if there is no server to revoke. As t is still incremented in this case, the adversary may query acceptance tokens to Revoke to progress system time if desired. The challenger also returns $\perp$ if the query is made during the challenge time q_t and $\overline{R}$ is not a subset of the current revocation list Q_{Rev} (including S_i as it is about to be revoked) – that is, if there exists a server other than S_i that is listed on $\overline{R}$ and so should be revoked at this challenge time, but is not currently revoked.

3.3.1 Public Verifiability

We extend the Public Verifiability game of Alderman et al. [2] to formalize in the HPVC model that no server may return an incorrect result for a computation without a verifier detecting it. We allow the adversary to corrupt other servers, generate arbitrary computations itself, and to perform verification steps himself.

Full Public Verifiability. This is captured in the game presented in Game 4. The game begins with the challenger setting up the system and running FnInit to initialize the challenge function F. The adversary, $\mathcal{A}$, is given the resulting public parameters and given oracle access to $\mathsf{FnInit}(\cdot, MK, PP)$, $\mathsf{Register}(\cdot, MK, PP)$, $\mathsf{Certify}(\cdot, (\cdot, \cdot), \cdot, \cdot, \cdot MK, PP)$ and $\mathsf{Revoke}(\cdot, (\cdot, \cdot), (\cdot, \cdot), MK, PP)$ as mentioned previously. All oracles simply run the relevant algorithm. Eventually, the adversary will finish this query phase and output challenge inputs for both objective and subjective policies, and both types of attribute set, which corresponds to $((\mathbb{O}^{\star}, \omega^{\star}, \mathbb{S}^{\star}, \psi^{\star}), l^{\star})$. The challenger will then generate a challenge by running ProbGen on this input $(\mathbb{S}^{\star}, \psi^{\star})$ for one of the modes, and give the resulting encoded input to $\mathcal{A}$. The adversary is again given oracle access and wins if it can produce an encoded output that verifies correctly but does not encode the value $F(x)$.

Selective, semi-static Public Verifiability. In Game 5 we present a selective, semi-static Public Verifiability game for HPVC.

The adversary first selects an input value. The challenger initializes a list of currently revoked entities Q_{Rev} and a time parameter t before running Setup and FnInit to create a public delegation key for the function F. The adversary is given the generated public parameters and must output a list $\overline{R}$ of servers to be revoked when the challenge is created. It is then given

Game 5 $\mathbf{Exp}_{\mathcal{A}}^{sSS-PubVerif}[\mathcal{HPVC}, \mathtt{mode}, F, q_t, 1^\kappa]$:

$((\mathbb{O}^\star, \omega^\star, \mathbb{S}^\star, \psi^\star), l^\star) \leftarrow \mathcal{A}(1^\kappa)$
$Q_{Rev} = \epsilon$, $t = 1$
$(PP, MK) \leftarrow \mathsf{Setup}(1^\kappa)$
$PK_F \leftarrow \mathsf{FnInit}(F, MK, PP)$
$\overline{R} \leftarrow \mathcal{A}(PK_F, PP)$
$\mathcal{A}^{\mathcal{O}}(PK_F, PP)$
if $(\overline{R} \not\subseteq Q_{Rev})$ **then return** 0
$(\sigma_{(\omega^\star,\mathbb{S}^\star)}, VK_{(\omega^\star,\mathbb{S}^\star)}, RK_{(\omega^\star,\mathbb{S}^\star)}) \leftarrow \mathsf{ProbGen}(\mathtt{mode}, (\omega^\star, \mathbb{S}^\star), l^\star, PK_F, PP)$
$\theta_{(\mathbb{O}^\star,\psi^\star),(\omega^\star,\mathbb{S}^\star)} \leftarrow \mathcal{A}^{\mathcal{O}}(\sigma_{(\omega^\star,\mathbb{S}^\star)}, VK_{(\omega^\star,\mathbb{S}^\star)}, RK_{(\omega^\star,\mathbb{S}^\star)})$
$(RT_{(\mathbb{O}^\star,\psi^\star),(\omega^\star,\mathbb{S}^\star)}, \tau_{(\mathbb{O}^\star,\psi^\star),(\omega^\star,\mathbb{S}^\star)}) \leftarrow \mathsf{BVerif}(\theta_{(\mathbb{O}^\star,\psi^\star),(\omega^\star,\mathbb{S}^\star)}, VK_{(\omega^\star,\mathbb{S}^\star)}, PP)$
$y_{(\mathbb{O}^\star,\psi^\star),(\omega^\star,\mathbb{S}^\star)} \leftarrow \mathsf{Retrieve}(RT_{(\mathbb{O}^\star,\psi^\star),(\omega^\star,\mathbb{S}^\star)}, \tau_{(\mathbb{O}^\star,\psi^\star),(\omega^\star,\mathbb{S}^\star)}, VK_{(\omega^\star,\mathbb{S}^\star)}, RK_{(\omega^\star,\mathbb{S}^\star)}, PP)$
if $((y_{(\mathbb{O}^\star,\psi^\star),(\omega^\star,\mathbb{S}^\star)}, \tau_{(\mathbb{O}^\star,\psi^\star),(\omega^\star,\mathbb{S}^\star)} \neq (\perp, (\mathsf{reject}, S_i)))$ **and** $(y_{(\mathbb{O}^\star,\psi^\star),(\omega^\star,\mathbb{S}^\star)} \neq F(x)))$ **then**
  return 1
else
  return 0

Oracle Query 1 $\mathcal{O}^{\mathsf{Certify}}(\mathtt{mode}, (\mathbb{O}, \psi), l, \mathcal{F}_i, S_i, MK, PP)$:

if $\mathtt{mode} = \mathrm{RPVC}$ **then**
  if $(F \in \mathcal{F}_i$ and $S_i \notin \overline{R})$ **or** $(t = q_t$ and $\overline{R} \not\subseteq Q_{\mathrm{Rev}} \setminus S_i)$ **then return** $\perp$
  $Q_{Rev} = Q_{Rev} \setminus S_i$
return $\mathsf{Certify}(\mathtt{mode}, (\mathbb{O}, \psi), l, \mathcal{F}_i, S_i, MK, PP)$;

Oracle Query 2 $\mathcal{O}^{\mathsf{Revoke}}(\tau_{(\mathbb{O},\psi),(\omega,\mathbb{S})}, (\mathbb{O}, \psi), (\omega, \mathbb{S}), MK, PP)$:

$t = t + 1$
if $\tau_{(\mathbb{O}^\star,\psi^\star),(\omega^\star,\mathbb{S}^\star)} = (\mathsf{accept}, \cdot)$ **then return** $\perp$
if $t = q_t$ and $\overline{R} \not\subseteq Q_{Rev} \cup S_i$ **then return** $\perp$
$Q_{Rev} = Q_{Rev} \cup S_i$;
return $\mathsf{Revoke}(\tau_{(\mathbb{O},\psi),(\omega,\mathbb{S})}, (\mathbb{O}, \psi), (\omega, \mathbb{S}), MK, PP)$;

oracle access to the above functions which simulate all values known to a real server as well as those learnt through corrupting entities. The challenger responds to Certify and Revoke queries as detailed in Oracle Queries 1 and 2 respectively. It must ensure that Q_{Rev} is kept up-to-date by adding or removing the queried entity, and in the case of revocation must increment the time parameter. It also ensures that issued keys will not lead to a trivial win.

Once the adversary has finished this query phase (and in particular, due to the parameterisation of the adversary, after exactly q_t Revoke queries), the challenger must check that the queries made by the adversary has indeed left the list of revoked entities to be at least that selected beforehand by the adversary. If there is a server that the adversary included on $\overline{R}$ but is not currently revoked, then the adversary loses the game. Otherwise, the challenger generates the challenge by running ProbGen on $x^\star$. The adversary is given the resulting encoded input and oracle access again, and wins the game if it creates an encoded output that verifies correctly yet does not encode the correct value $F(x)$.

Definition 8. *The* advantage *of a PPT adversary $\mathcal{A}$ making a polynomial number of queries q (including q_t Revoke queries) is defined as:*

- $Adv_{\mathcal{A}}^{PubVerif}(\mathcal{HPVC}, F, 1^\kappa, q) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{PubVerif}[\mathcal{HPVC}, F, 1^\kappa] = 1]$
- $Adv_{\mathcal{A}}^{sSS\text{-}PubVerif}(\mathcal{HPVC}, F, 1^\kappa, q) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{sSS\text{-}PubVerif}[\mathcal{HPVC}, F, q_t, 1^\kappa] = 1]$

A HPVC is secure against Game PubVerif or sSS-PubVerif *for a function F, if for all PPT adversaries $\mathcal{A}$,*

$$Adv_{\mathcal{A}}^{PubVerif,sSS\text{-}PubVerif}(\mathcal{HPVC},\ F,\ 1^{\kappa},q) \leq \text{negl}(\kappa).$$

3.3.2 Blind Verification

In Game 6 we capture the notion of Blind Verification in the HPVC setting. We require that a verifier that does not hold the output retrieval key may not learn the result of the computation, even though they hold the public verification key. As in Game 3, the challenger will select an input x from the domain of F. We then embed x as the challenge input data according to the mode of operation (we have used here the notation s and P for the dummy attribute set and policy respectively, as this then permits the access control policies from Section 4 as well). The adversary is given the results of a computation on this data and must determine the value of $F(x)$. As in Section 2.2.2, we choose not to provide the adversary with the encoded inputs (ciphertexts) or the evaluation key to avoid giving them the descriptive label that could allow them to learn the input data.

Game 6 $\mathbf{Exp}_{\mathcal{A}}^{BVerif}[\mathcal{HPVC}, \texttt{mode}, F, 1^{\kappa}]$:

1: $(PP, MK) \leftarrow \mathsf{Setup}(1^{\kappa})$
2: $PK_F \leftarrow \mathsf{FnInit}(F, MK, PP)$
3: $x \xleftarrow{\$} \mathsf{Dom}(F)$
4: $S_i \xleftarrow{\$} \mathcal{U}_{ID}$
5: **if** mode = VDC **then**
6: $\quad \psi = x$, $\mathbb{S} = F$, $\mathbb{O} = P$, $\omega = s$, $l = l(x)$
7: **else**
8: $\quad \omega = x$, $\mathbb{O} = F$, $\mathbb{S} = P$, $\psi = s$, $l = l(F)$
9: $SK_{S_i} \leftarrow \mathsf{Register}(S_i, MK, PP)$
10: $EK_{(\mathbb{O},\psi),S_i} \leftarrow \mathsf{Certify}(\texttt{mode}, (\mathbb{O}, \psi), l, \{F\}, S_i, MK, PP)$
11: $(\sigma_{(\omega,\mathbb{S})}, VK_{(\omega,\mathbb{S})}, RK_{(\omega,\mathbb{S})}) \leftarrow \mathsf{ProbGen}(\texttt{mode}, (\omega, \mathbb{S}), l, PK_F, PP)$
12: $\theta_{(\mathbb{O},\psi),(\omega,\mathbb{S})} \leftarrow \mathsf{Compute}(\texttt{mode}, \sigma_{(\omega,\mathbb{S})}, EK_{(\mathbb{O},\psi),S_i}, SK_{S_i}, PP)$
13: $\hat{y} \leftarrow \mathcal{A}^{\mathcal{O}}(\theta_{(\mathbb{O},\psi),(\omega,\mathbb{S})}, VK_{(\omega,\mathbb{S})}, PK_F, PP)$
14: **if** $(\hat{y} = F(x))$ **then**
15: $\quad$ **return** 1
16: **else**
17: $\quad$ **return** 0

Definition 9. *The* advantage *of a PPT adversary $\mathcal{A}$ making a polynomial number of queries q in the Blind Verification Experiment is defined as:*

$$Adv_{\mathcal{A}}^{BVerif}(\mathcal{HPVC}, F, 1^{\kappa}, q) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{BVerif}[\mathcal{HPVC}, F, 1^{\kappa}] = 1] - \max_{y \in \mathsf{Ran}(F)} (\Pr_{x \in \mathsf{Dom}(F)}[F(x) = y]).$$

A RPVC is secure against blind verification *for a function F, if for all PPT adversaries $\mathcal{A}$,*

$$Adv_{\mathcal{A}}^{BVerif}(\mathcal{HPVC},\ F,\ 1^{\kappa},q) \leq \text{negl}(\kappa).$$

3.3.3 Revocation

The notion of revocation requires that if a server is detected as misbehaving, i.e. the BVerif algorithm output $\tau_{(\mathbb{O},\psi),(\omega,\mathbb{S})} = (\mathsf{reject}, S_i)$, then any subsequent evaluations by S_i should be rejected. The motivation here is that even though we have outsourced the costly computation and pre-processing stages to the server and KDC respectively, there is still a cost involved in delegating and verifying a computation. If a server is known not to be trustworthy then we remove any incentive for it to attempt to provide an outsourcing service. In addition, we may want to punish and further disincentivize malicious servers by removing their ability to perform work for a period of time. Finally, also from a privacy perspective, we may not wish to supply input data or specific queries on server's databases depending on the mode (e.g. in VDC, we may wish to prevent a malicious server convincing users of a particular, false, property of their remote data).

Game 7 $\mathbf{Exp}_{\mathcal{A}}^{Revocation}[\mathcal{HPVC}, \mathtt{mode}, F, 1^{\kappa}]$:

$\mathtt{chall} = \mathtt{false}$
$Q_{\mathrm{Rev}} = \epsilon$
$(PP, MK) \leftarrow \mathsf{Setup}(1^{\kappa})$
$PK_F \leftarrow \mathsf{FnInit}(F, MK, PP)$
$((\mathbb{O}^{\star}, \omega^{\star}, \mathbb{S}^{\star}, \psi^{\star}), l^{\star}) \leftarrow \mathcal{A}^{\mathcal{O}}(PK_F, PP)$
$\mathtt{chall} = \mathtt{true}$
$(\sigma_{(\omega^{\star},\mathbb{S}^{\star})}, VK_{(\omega^{\star},\mathbb{S}^{\star})}, RK_{(\omega^{\star},\mathbb{S}^{\star})}) \leftarrow \mathsf{ProbGen}(\mathtt{mode}, (\omega^{\star}, \mathbb{S}^{\star}), l^{\star}, PK_F, PP)$
$\theta_{(\mathbb{O}^{\star},\psi^{\star}),(\omega^{\star},\mathbb{S}^{\star})} \leftarrow \mathcal{A}^{\mathcal{O}}(\sigma_{(\omega^{\star},\mathbb{S}^{\star})}, VK_{(\omega^{\star},\mathbb{S}^{\star})}, RK_{(\omega^{\star},\mathbb{S}^{\star})})$
$(RT_{(\mathbb{O}^{\star},\psi^{\star}),(\omega^{\star},\mathbb{S}^{\star})}, \tau_{(\mathbb{O}^{\star},\psi^{\star}),(\omega^{\star},\mathbb{S}^{\star})}) \leftarrow \mathsf{BVerif}(\theta_{(\mathbb{O}^{\star},\psi^{\star}),(\omega^{\star},\mathbb{S}^{\star})}, VK_{(\omega^{\star},\mathbb{S}^{\star})}, PP)$
$y_{(\mathbb{O}^{\star},\psi^{\star}),(\omega^{\star},\mathbb{S}^{\star})} \leftarrow \mathsf{Retrieve}(RT_{(\mathbb{O}^{\star},\psi^{\star}),(\omega^{\star},\mathbb{S}^{\star})}, \tau_{(\mathbb{O}^{\star},\psi^{\star}),(\omega^{\star},\mathbb{S}^{\star})}, VK_{(\omega^{\star},\mathbb{S}^{\star})}, RK_{(\omega^{\star},\mathbb{S}^{\star})}, PP)$
if $(\tau_{(\mathbb{O}^{\star},\psi^{\star}),(\omega^{\star},\mathbb{S}^{\star})} = (\mathsf{accept}, S_i))$ **and** $(S_i \in Q_{\mathrm{Rev}})$ **then**
    return 1
else
    return 0

Oracle Query 3 $\mathcal{O}^{\mathsf{Certify}}(\mathtt{mode}, (\mathbb{O}, \psi), l, \mathcal{F}_i, S_i, MK, PP)$:

if $\mathtt{chall} = \mathtt{false}$ **then** $Q_{\mathrm{Rev}} = Q_{\mathrm{Rev}} \setminus S_i$
return $\mathsf{Certify}(\mathtt{mode}, (\mathbb{O}, \psi), l, \mathcal{F}_i, S_i, MK, PP)$

Oracle Query 4 $\mathcal{O}^{\mathsf{Revoke}}(\tau_{(\mathbb{O},\psi),(\omega,\mathbb{S})}, (\mathbb{O}, \psi), (\omega, \mathbb{S}), MK, PP)$:

$r \leftarrow \mathsf{Revoke}(\tau_{(\mathbb{O},\psi),(\omega,\mathbb{S})}, (\mathbb{O}, \psi), (\omega, \mathbb{S}), MK, PP)$;
if $r \neq \perp$ and $\mathtt{chall} = \mathtt{false}$ **then** $Q_{\mathrm{Rev}} = Q_{\mathrm{Rev}} \cup S_i$;
return r;

Full Revocation. The full notion, in Game 7, begins by declaring a Boolean flag `chall` which is initially set to `false` and a list Q_{Rev} which servers will be added to when revoked and removed from when certified. The `chall` flag will be set to `true` when the challenge is created, and after this point Q_{Rev} is no longer updated. Thus Q_{Rev} will comprise all servers that are revoked at the challenge time and hence all servers that, if an adversary can output a result 'from' one of these servers and have it accepted, will count as a win for the adversary.

The game proceeds in a similar fashion to Public Verifiability with the challenger running Setup and FnInit to initialize the system and providing the public parameters to the adversary along with oracle access. All oracles simply run the relevant algorithms except for Certify and Revoke which additionally maintain the list of revoked entities as mentioned above and specified in Oracle Queries 3 and 4 respectively. After the adversary has finished this query phase, it outputs a challenge input for both objective and subjective policies and both types of attribute set, and the challenger sets the `chall` flag to `true`. It then generates the challenge by running ProbGen and gives the resulting parameters to the adversary along with oracle access again (however, since `chall` is set, Q_{Rev} will no longer be updated). Eventually, the adversary outputs a result $\theta_{(\mathbb{O}^{\star},\psi^{\star}),(\omega^{\star},\mathbb{S}^{\star})}$ and wins if Verify outputs accept for a server that was revoked when the challenge was generated (even a correct result).

Selective, semi-static Revocation. On the other hand, the selective, semi-static notion of Revocation given in Game 8 proceeds exactly as the sSS-PubVerif game for Public Verifiability except for the winning condition. Here, the adversary wins if it outputs *any* result (even a correct encoding of $F(x)$) that is accepted as a valid response from any server that was revoked at the time of the challenge. This game also uses the Certify and Revoke oracles specified in Oracle Queries 1 and 2 respectively.

Definition 10. *The* advantage *of a PPT adversary $\mathcal{A}$ making a polynomial number of queries q (including q_t* Revoke *queries) against Revocation or sSS-Revocation is defined as:*

Game 8 $\mathbf{Exp}_{\mathcal{A}}^{sSS-Revocation}[\mathcal{HPVC}, \mathtt{mode}, F, q_t, 1^\kappa]$:

$((\mathbb{O}^\star, \omega^\star, \mathbb{S}^\star, \psi^\star), l^\star) \leftarrow \mathcal{A}(1^\kappa)$
$Q_{Rev} = \epsilon$, $t = 1$
$(PP, MK) \leftarrow \mathsf{Setup}(1^\kappa)$
$PK_F \leftarrow \mathsf{FnInit}(F, MK, PP)$
$\overline{R} \leftarrow \mathcal{A}(PK_F, PP)$
$\mathcal{A}^{\mathcal{O}}(PK_F, PP)$
if $(\overline{R} \not\subseteq Q_{Rev})$ **then return** 0
$(\sigma_{(\omega^\star,\mathbb{S}^\star)}, VK_{(\omega^\star,\mathbb{S}^\star)}, RK_{(\omega^\star,\mathbb{S}^\star)}) \leftarrow \mathsf{ProbGen}(\mathtt{mode}, (\omega^\star, \mathbb{S}^\star), l^\star, PK_F, PP)$
$\theta_{(\mathbb{O}^\star,\psi^\star),(\omega^\star,\mathbb{S}^\star)} \leftarrow \mathcal{A}^{\mathcal{O}}(\sigma_{(\omega^\star,\mathbb{S}^\star)}, VK_{(\omega^\star,\mathbb{S}^\star)}, RK_{(\omega^\star,\mathbb{S}^\star)})$
$(RT_{(\mathbb{O}^\star,\psi^\star),(\omega^\star,\mathbb{S}^\star)}, \tau_{(\mathbb{O}^\star,\psi^\star),(\omega^\star,\mathbb{S}^\star)}) \leftarrow \mathsf{BVerif}(\theta_{(\mathbb{O}^\star,\psi^\star),(\omega^\star,\mathbb{S}^\star)}, VK_{(\omega^\star,\mathbb{S}^\star)}, PP)$
$y_{(\mathbb{O}^\star,\psi^\star),(\omega^\star,\mathbb{S}^\star)} \leftarrow \mathsf{Retrieve}(RT_{(\mathbb{O}^\star,\psi^\star),(\omega^\star,\mathbb{S}^\star)}, \tau_{(\mathbb{O}^\star,\psi^\star),(\omega^\star,\mathbb{S}^\star)}, VK_{(\omega^\star,\mathbb{S}^\star)}, RK_{(\omega^\star,\mathbb{S}^\star)}, PP)$
if $(\tau_{(\mathbb{O}^\star,\psi^\star),(\omega^\star,\mathbb{S}^\star)} = (\mathsf{accept}, S_i))$ **and** $(S_i \in \overline{R})$ **then**
  return 1
else
  return 0

- $Adv_{\mathcal{A}}^{Revocation}(\mathcal{HPVC}, F, 1^\kappa, q) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{Revocation}[\mathcal{HPVC}, F, 1^\kappa] = 1]$
- $Adv_{\mathcal{A}}^{sSS\text{-}Revocation}(\mathcal{HPVC}, F, 1^\kappa, q) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{sSS\text{-}Revocation}[\mathcal{HPVC}, F, q_t, 1^\kappa] = 1]$

A HPVC is secure against Revocation or sSS-Revocation *for a function F, if for all PPT adversaries $\mathcal{A}$,*

$$Adv_{\mathcal{A}}^{Revocation, sSS\text{-}Revocation}(\mathcal{HPVC},\ F,\ 1^\kappa, q) \leq \mathrm{negl}(\kappa).$$

3.4 Instantiation

We construct a HPVC scheme for a family $\mathcal{F}$ of monotone Boolean formulas closed under complement using Dual-Policy ABE in a black-box manner. Consider a function to be delegated $F : \{0,1\}^n \to \{0,1\}$ and its complement function $\overline{F} = F(x) \oplus 1$As in Section 2.3, n-bit binary input strings are encoded as attribute sets $x \subseteq \mathcal{U}_x$. Let $\mathcal{U}_l$ be a set of attributes (disjoint from $\mathcal{U}_x$) uniquely labelling each function and possible input, and let $\mathcal{U}_{ID}$ represent server identities. Let g be a one-way function and $\mathcal{DPABE}$ = (DPABE.Setup, DPABE.Encrypt, DPABE.KeyGen, DPABE.KeyUpdate, DPABE.Decrypt) be a revocable DP-ABE scheme for $\mathcal{F}$ (see Appendix B) with attribute universe $\mathcal{U} = \mathcal{U}_x \cup \mathcal{U}_l \cup \mathcal{U}_{ID} \cup T_O \cup T_S$, where T_O and T_S are extra attributes that enable single-policy modes [6].

These allow a DP-ABE scheme to efficiently function as either KP-ABE or CP-ABE. For KP-ABE, the subjective policy $\mathbb{S} = \{\{T_S\}\}$ is satisfied by the presence in ψ of the special attribute T_S – thus, $\mathbb{S}$ is always trivially satisfied and decryption only depends on the objective attributes and policy. Similarly, for CP-ABE, $\omega = \{T_O\}$ and $\mathbb{O} = \{\{T_O\}\}$. We will initialize two independent DP-ABE systems over $\mathcal{U}$. Hence, we define a total of four additional attributes: T_O^0, T_S^0 relating to the first system, and T_O^1, T_S^1 for the second system. We denote the complement functions in different modes as follows: In RPVC, $\mathbb{O} = F$ and $\mathbb{S} = \{\{T_S^0\}\}$. Hence $\overline{\mathbb{O}} = \overline{F}$ whilst $\overline{\mathbb{S}} = \{\{T_S^1\}\}$. Similarly, for VDC, $\overline{\mathbb{O}} = \{\{T_0^1\}\}$ and $\overline{\mathbb{S}} = \overline{F}$.

Overview. Each mode operates by encrypting a pair of random messages and issuing keys such that the recovery of one message implies whether the ciphertext was linked to F or $\overline{F}$ and hence if $F(x) = 1$ or 0. Ciphertext indistinguishability ensures an adversary cannot cheat by returning the other message. Setup initializes two revocable DP-ABE schemes over the universe$\mathcal{U}$, an empty two-dimensional array L_{Reg} (as in the VDC construction above), a list of revoked servers and a time source $\mathbb{T}$ (e.g. a networked clock or counter updated by Revoke) to

index update keys. FnInit sets the public delegation key PK_F to be the public parametersfor the system (since we use public key primitives).

Register runs a signature KeyGen algorithm and adds the verification key to $L_{Reg}[S_i][0]$. Signatures ensure that honest servers are not impersonated and maliciously revoked. Certify first adds a pair to $L_{Reg}[S_i][1]$ comprising the function identifier and the data label l for each function $F_j \in \mathcal{F}_i$ chosen by the server, and removes S_i from the revocation list. It then runs KeyGen for the first DP-ABE system to generate a decryption key for $(\mathbb{O} \wedge l, \psi \cup l)$. The inclusion of the label $l \in \mathcal{U}_l$, as an additional attribute or as a conjunctive policy clause, ensures that only matching keys and ciphertexts may be used (i.e. that a key for different data may not be used). The KDC should check that the label actually corresponds to the input (for example, define a one-way, injective label mapping from $\mathcal{F} \times \mathcal{U}_x$ to $\mathcal{U}_l$). It also generates an update key for the current time period to prove that S_i is not currently revoked. If operating in RPVC mode, another pair of keys is generated using the second DP-ABE system and for the complement inputs.

ProbGen chooses random messages m_0 and m_1. A random bit b randomly permutes the messages such that a verifier that does not know $b$does not know which message was recovered and hence the value of $F(x)$. Message m_b is encrypted with $(\omega \cup l, \mathbb{S} \wedge l)$ and the first system parameters, whilst m_{1-b} is encrypted using the complement policy and either the first system parameters for VDC or the second for RPVC (the attribute set remains the same as it is either the same input data x in RPVC, or the same special attribute T_O^0 in VDC). The verification key is computed by applying the one-way function g to the messages (the one-wayness allows the key to be published), and b forms the output retrieval key(as this reveals the order of the decrypted results and hence the output).

Compute decrypts the two ciphertexts and signs the results, again ensuring that the different modes use the correct system parameters. BVerif verifies the signature and compares the components of the verification key against g applied to the decryptions. If either matches (i.e. the server successfully recovered a message), that plaintext is returned as the retrieval token and the output token is accept. Otherwise the result is rejected and the server is reported for revocation. Retrieve checks which message was returned correctly and hence the result of the computation – m_0 was encrypted for the non-complemented input set so implies $F(x) = 1$. Finally, to revoke a server, the KDC refreshes the time source (e.g. increments a counter) and generates new update keys for all unrevoked entities in the system. More formally, our scheme is defined by Algorithms 9–17.

Alg. 9 $(PP, MK) \leftarrow \mathsf{HPVC.Setup}(1^\kappa)$

1: $(MPK^0_{\text{ABE}}, MSK^0_{\text{ABE}}, T^0_O, T^0_S) \leftarrow \mathsf{DPABE.Setup}(1^\kappa, \mathcal{U})$
2: $(MPK^1_{\text{ABE}}, MPK^1_{\text{ABE}}, T^1_O, T^1_S) \leftarrow \mathsf{DPABE.Setup}(1^\kappa, \mathcal{U})$
3: **for** $S_i \in \mathcal{U}_{\text{ID}}$ **do**
4: $\quad L_{\text{Reg}}[S_i][0] = \epsilon$, $L_{\text{Reg}}[S_i][1] = \{\epsilon\}$
5: Initialize $\mathbb{T}$, $L_{\text{Rev}} = \epsilon$
6: $PP = (MPK^0_{\text{ABE}}, MPK^1_{\text{ABE}}, L_{\text{Reg}}, T^0_O, T^1_O, T^0_S, T^1_S, \mathbb{T})$
7: $MK = (MSK^0_{\text{ABE}}, MSK^1_{\text{ABE}}, L_{\text{Rev}})$

Alg. 10 $PK_F \leftarrow \mathsf{HPVC.FnInit}(F, MK, PP)$

1: Set $PK_F = PP$

Alg. 11 $SK_{S_i} \leftarrow \mathsf{HPVC.Register}(S_i, MK, PP)$

$(SK_{\mathrm{Sig}}, VK_{\mathrm{Sig}}) \leftarrow \mathsf{Sig.KeyGen}(1^{\kappa})$
$SK_{S_i} = SK_{\mathrm{Sig}}$
$L_{\mathrm{Reg}}[S_i][0] = L_{\mathrm{Reg}}[S_i][0] \cup VK_{\mathrm{Sig}}$

Alg. 12 $EK_{(\mathbb{O},\psi),S_i} \leftarrow \mathsf{HPVC.Certify}(S_i, \mathtt{mode}, (\mathbb{O},\psi), l, \mathcal{F}_i, MK, PP)$

for $F_j \in \mathcal{F}_i$ **do**
    $L_{\mathrm{Reg}}[S_i][1] = L_{\mathrm{Reg}}[S_i][1] \cup (F_j, l)$
$L_{\mathrm{Rev}} = L_{\mathrm{Rev}} \setminus S_i$, $t \leftarrow \mathbb{T}$
$SK^0_{\mathrm{ABE}} \leftarrow \mathsf{DPABE.KeyGen}(S_i, (\mathbb{O} \wedge l, \psi \cup l), MSK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}})$
$UK^0_{L_{\mathrm{Rev}},t} \leftarrow \mathsf{DPABE.KeyUpdate}(L_{\mathrm{Rev}}, t, MSK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}})$
if $\mathtt{mode} = RPVC$ **then**
    $SK^1_{\mathrm{ABE}} \leftarrow \mathsf{DPABE.KeyGen}(S_i, (\overline{\mathbb{O}} \wedge l, \overline{\psi} \cup l), MSK^1_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}})$
    $UK^1_{L_{\mathrm{Rev}},t} \leftarrow \mathsf{DPABE.KeyUpdate}(L_{\mathrm{Rev}}, t, MSK^1_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}})$
else $SK^1_{\mathrm{ABE}} = \perp$, $UK^1_{L_{\mathrm{Rev}},t} = \perp$
$EK_{(\mathbb{O},\psi),S_i} = (SK^0_{\mathrm{ABE}}, SK^1_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t}, UK^1_{L_{\mathrm{Rev}},t})$

Alg. 13 $(\sigma_{(\omega,\mathbb{S})}, VK_{(\omega,\mathbb{S})}, RK_{(\omega,\mathbb{S})}) \leftarrow \mathsf{HPVC.ProbGen}(\mathtt{mode}, (\omega,\mathbb{S}), l, PK_F, PP)$

$(m_0, m_1) \xleftarrow{\$} \mathcal{M} \times \mathcal{M}$, $b \xleftarrow{\$} \{0,1\}$, $t \leftarrow \mathbb{T}$
$c_b \leftarrow \mathsf{DPABE.Encrypt}(t, (\omega \cup l, \mathbb{S} \wedge l), m_b, MPK^0_{\mathrm{ABE}})$
if $\mathtt{mode} = VDC$ **then**
    $c_{1-b} \leftarrow \mathsf{DPABE.Encrypt}(t, (\omega \cup l, \bar{\mathbb{S}} \wedge l), m_{1-b}, MPK^0_{\mathrm{ABE}})$
else $c_{1-b} \leftarrow \mathsf{DPABE.Encrypt}(t, (\omega \cup l, \bar{\mathbb{S}} \wedge l), m_{1-b}, MPK^1_{\mathrm{ABE}})$
return $\sigma_{\omega,\mathbb{S}} = (c_b, c_{1-b})$, $VK_{\omega,\mathbb{S}} = (g(m_b), g(m_{1-b}), L_{\mathrm{Reg}})$ and $RK_{\omega,\mathbb{S}} = b$

Alg. 14 $\theta_{(\mathbb{O},\psi),(\omega,\mathbb{S})} \leftarrow \mathsf{HPVC.Compute}(\mathtt{mode}, \sigma_{(\omega,\mathbb{S})}, EK_{(\mathbb{O},\psi),S_i}, SK_{S_i}, PP)$

Parse $EK_{(\mathbb{O},\psi),S_i} = (SK^0_{\mathrm{ABE}}, SK^1_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t}, UK^1_{L_{\mathrm{Rev}},t})$ and $\sigma_{\omega,\mathbb{S}} = (c, c')$
$d_b \leftarrow \mathsf{DPABE.Decrypt}(c, SK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t})$
if $\mathtt{mode} = VDC$ **then**
    $d_{1-b} \leftarrow \mathsf{DPABE.Decrypt}(c', SK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t})$
else $d_{1-b} \leftarrow \mathsf{DPABE.Decrypt}(c', SK^1_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}}, UK^1_{L_{\mathrm{Rev}},t})$
$\gamma \leftarrow \mathsf{Sig.Sign}((d_b, d_{1-b}, S_i), SK_{S_i})$
$\theta_{(\omega,\mathbb{S}),(\mathbb{O},\psi)} = (d_b, d_{1-b}, S_i, \gamma)$

Alg. 15 $(RT_{(\mathbb{O},\psi),(\omega,\mathbb{S})}, \tau_{(\mathbb{O},\psi),(\omega,\mathbb{S})}) \leftarrow \mathsf{HPVC.BVerif}(\theta_{(\mathbb{O},\psi),(\omega,\mathbb{S})}, VK_{(\omega,\mathbb{S})}, PP)$

Parse $VK_{(\omega,\mathbb{S})} = (VK, VK', L_{\mathrm{Reg}})$ and $\theta_{(\mathbb{O},\psi),(\omega,\mathbb{S})} = (d, d', S_i, \gamma)$
if $L_{\mathrm{Reg}}[S_i][0] \neq \epsilon$ **and** $(F_j, l) \in L_{\mathrm{Reg}}[S_i][1]$ **then**
    if $\mathsf{accept} \leftarrow \mathsf{Sig.Verify}((d, d', S_i), \gamma, L_{\mathrm{Reg}}[S_i][0])$ **then**
        if $VK = g(d)$ **then return** $(RT_{(\mathbb{O},\psi),(\omega,\mathbb{S})} = d, \tau_{(\mathbb{O},\psi),(\omega,\mathbb{S})} = (\mathsf{accept}, S))$
        else if $VK' = g(d')$ **then return** $(RT_{(\mathbb{O},\psi),(\omega,\mathbb{S})} = d', \tau_{(\mathbb{O},\psi),(\omega,\mathbb{S})} = (\mathsf{accept}, S))$
        else return $(RT_{(\mathbb{O},\psi),(\omega,\mathbb{S})} = \perp, \tau_{(\mathbb{O},\psi),(\omega,\mathbb{S})} = (\mathsf{reject}, S))$
return $(RT_{(\mathbb{O},\psi),(\omega,\mathbb{S})} = \perp, \tau_{(\mathbb{O},\psi),(\omega,\mathbb{S})} = (\mathsf{reject}, \perp))$

Alg. 16 $y_{(\mathbb{O},\psi),(\omega,\mathbb{S})} \leftarrow \mathsf{HPVC.Retrieve}(RT_{(\mathbb{O},\psi),(\omega,\mathbb{S})}, \tau_{(\mathbb{O},\psi),(\omega,\mathbb{S})}, VK_{(\omega,\mathbb{S})}, RK_{(\omega,\mathbb{S})}, PP)$

Parse $VK_{(\omega,\mathbb{S})} = (g(m_b), g(m_{1-b}), L_{\mathrm{Reg}})$, $\theta_{(\mathbb{S},\omega),(\mathbb{O},\psi)} = (d_b, d_{1-b}, S_i, \gamma)$, $RK_{(\omega,\mathbb{S})} = b$, and $(RT_{(\mathbb{O},\psi),(\omega,\mathbb{S})}, \tau_{(\mathbb{O},\psi),(\omega,\mathbb{S})})$ where $RT_{(\mathbb{O},\psi),(\omega,\mathbb{S})} \in \{d_b, d_{1-b}, \perp\}$
if $\tau_{(\mathbb{O},\psi),(\omega,\mathbb{S})} = (\mathsf{accept}, S)$ and $g(RT_{(\mathbb{O},\psi),(\omega,\mathbb{S})}) = g(m_0)$ **then return** $y_{(\mathbb{O},\psi),(\omega,\mathbb{S})} = 1$
if $\tau_{(\mathbb{O},\psi),(\omega,\mathbb{S})} = (\mathsf{accept}, S)$ and $g(RT_{(\mathbb{O},\psi),(\omega,\mathbb{S})}) = g(m_1)$ **then return** $y_{(\mathbb{O},\psi),(\omega,\mathbb{S})} = 0$
return $y_{(\mathbb{O},\psi),(\omega,\mathbb{S})} = \perp$

Alg. 17 $\{EK_{(\mathbb{O},\psi),S'}\}$ or $\perp \leftarrow \mathsf{HPVC.Revoke}(\tau_{(\mathbb{O},\psi),(\omega,\mathbb{S})}, (\mathbb{O},\psi), (\omega,\mathbb{S}), MK, PP)$

1: **if** $\tau_{(\mathbb{O},\psi),(\omega,\mathbb{S})} \neq (\mathsf{reject}, S_i)$ **then return** $\perp$
2: $L_{\mathrm{Reg}}[S][1] = \{\epsilon\}$, $L_{\mathrm{Rev}} = L_{\mathrm{Rev}} \cup S_i$
3: Refresh $\mathbb{T}$, $t \leftarrow \mathbb{T}$
4: $UK^0_{L_{\mathrm{Rev}},t} \leftarrow \mathsf{DPABE.KeyUpdate}(L_{\mathrm{Rev}}, t, MSK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}})$
5: $UK^1_{L_{\mathrm{Rev}},t} \leftarrow \mathsf{DPABE.KeyUpdate}(L_{\mathrm{Rev}}, t, MSK^1_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}})$
6: **for** all $S' \in \mathcal{U}_{\mathrm{ID}}$ **do**
7: Parse $EK_{(\mathbb{O},\psi),S'}$ as $(SK^0_{\mathrm{ABE}}, SK^1_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t-1}, UK^1_{L_{\mathrm{Rev}},t-1})$
8: Update and send $EK_{(\mathbb{O},\psi),S'} = (SK^0_{\mathrm{ABE}}, SK^1_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t}, UK^1_{L_{\mathrm{Rev}},t})$

Theorem 2. *Given a secure* IND-sHRSS rkDPABE *scheme for a class of Boolean functions $\mathcal{F}$ closed under complement, a one-way function g, and a signature scheme secure against* EUF-CMA*, then $\mathcal{HPVC}$ is secure in the sense of selective semi-static Public Verifiability, Blind Verification and selective semi-static Revocation.*

Informally, Public Verifiability and Revocation reduce to the indistinguishability of ciphertexts within the rkDPABE scheme which allows us to replace the message for the unsatisfied function (which cannot be decrypted) with the challenge for a one-way function game. Then an adversary against these games can attack the verification token for this message. Blind Verification relies on a standard probability argument.

3.5 Security Proofs

In this section we provide the proofs of security in order to prove Theorem 2.

The proof of Blind Verification proceeds exactly as for VDC in Lemma 2 in Section 2.2.2 with updated notation.

3.5.1 Proof of Public Verifiability

Lemma 3. *The HPVC construction defined by Algorithms 9–16 is secure against Public Verifiability (Game 5) under the same assumptions as in Theorem 2.*

One way to view this proof is to reduce, based on the `mode` to either the RPVC or VDC Public Verifiability game. Observe that the choice of dummy attributes and a dummy policy that is trivially satisfied by the presence of that dummy attribute means that one type of policy is always trivially satisfied, and hence decryption hinges entirely on the remaining policy. Thus we are in the setting of the single mode games.

We now give a proof sketch of a unified proof.

Proof. Let $\mathcal{A}_{HPVC}$ be an adversary with non-negligible advantage against the selective, semi-static Public Verifiability game (Game 5) when instantiated with Algorithms 9–16. We define the following three games:

- **Game 0.** This is the correct selective, semi-static Public Verifiability game as in Game 5.
- **Game 1.** This differs from **Game 0** in that ProbGen no longer returns an encryption of m_0 and m_1. Instead, we choose a random message $m' \neq m_0, m_1$. Note that there are two ciphertexts created during ProbGen (being the encryption of m_0 and m_1) and that one of these will be associated with the function F, and the other with the complement function $\overline{F}$. Now, only one of F and $\overline{F}$ will be satisfied by the input data $x_i^\star$. We replace the plaintext associated with the *unsatisfied* function by m' which is unrelated to m_0, m_1 and the verification keys.

- **Game 2.** This is the same as **Game 1** with the exception that instead of choosing a random message m', we implicitly set m' to be the challenge input w in the one-way function game.

By hopping from **Game 0** to **Game 2**, we show that an adversary with non-negligible advantage against Public Verifiability can be used to construct an adversary that inverts the one-way function g.

Game 0 to Game 1. We begin by showing that there is a negligible distinguishing advantage between **Game 0** and **Game 1**. Suppose otherwise, that $\mathcal{A}_{HPVC}$ can distinguish the two games with non-negligible advantage δ. We construct an adversary $\mathcal{A}_{ABE}$ that uses $\mathcal{A}_{HPVC}$ as a subroutine to break the IND-sHRSS security of the rkDPABE scheme (Game 10). We consider a challenger $\mathcal{C}$ playing the IND-sHRSS game with $\mathcal{A}_{ABE}$, who in turn acts as a challenger in the selective, semi-static Public Verifiability game for $\mathcal{A}_{HPVC}$:

1. $\mathcal{A}_{HPVC}$ is given the security parameter, and declares its choice of challenge input parameters $(\mathbb{O}^\star, \omega^\star, \mathbb{S}^\star, \psi^\star)$ and the challenge label $l^\star$.

2. $\mathcal{A}_{ABE}$ must send a challenge input $(\tilde{\omega}, \tilde{\mathbb{S}})$ and a challenge time period $\tilde{t}$ to the challenger. It first sets $\tilde{t} = q_t$. Observe that in VDC mode, $\mathbb{S}^\star$ corresponds to the function F and $\psi^\star$ is the challenge input data $x_i^\star$. In RPVC mode, $\mathbb{O}^\star = F$ and ω corresponds to the challenge input $x_i^\star$. The other inputs are either dummy attributes or corresponding policies, and these policies are trivially satisfied by the dummy attribute. Now, using the relevant inputs, $\mathcal{A}_{ABE}$ computes $r = F(x_{i^\star})$.

   - If the challenge mode is RPVC, set $\tilde{\omega} = \omega^\star \cup l^\star = x_i^\star \cup l(F)$ and $\tilde{\mathbb{S}} = \mathbb{S}^\star \wedge l^\star = \{\{T_S^0\}\} \wedge l(F)$.
   - If the challenge mode is VDC, we want to set $(\tilde{\omega}, \tilde{\mathbb{S}})$ such that the pair is not satisfied by the challenge input.
     - If $r = 1$: $\tilde{\omega} = \omega^\star \cup l^\star = \{T_O^0\} \cup l(x_i^\star)$ and $\tilde{\mathbb{S}} = \overline{\mathbb{S}^\star} \wedge l^\star = \overline{F} \wedge l(x_i^\star)$
     - If $r = 0$: $\tilde{\omega} = \omega^\star \cup l^\star = \{T_O^0\} \cup l(x_i^\star)$ and $\tilde{\mathbb{S}} = \mathbb{S}^\star \wedge l^\star = F \wedge l(x_i^\star)$

3. $\mathcal{C}$ runs the DPABE.Setup algorithm on the security parameter to generate MPK_{ABE} and MSK_{ABE}. He gives MPK_{ABE} to $\mathcal{A}_{ABE}$.

4. $\mathcal{A}_{ABE}$ initializes Q_{Rev} and t and simulates running HPVC.Setup such that the outcome is consistent with MPK_{ABE}. It runs lines 3 to 5 as written, sets $MPK_{\text{ABE}}^0 = MPK_{\text{ABE}}$ as given by $\mathcal{C}$, and implicitly sets $MSK_{\text{ABE}}^0 = MSK_{\text{ABE}}$. It also runs DPABE.Setup himself to generate a second DP-ABE system.

5. $\mathcal{A}_{ABE}$ runs HPVC.FnInit as written and gives the resulting PK_F and the public parameters to $\mathcal{A}_{HPVC}$. $\mathcal{A}_{HPVC}$ returns a list $\overline{R}$ of servers to be revoked at the time of the challenge, which $\mathcal{A}_{ABE}$ forwards to $\mathcal{C}$.

6. $\mathcal{A}_{HPVC}$ is now given oracle access which $\mathcal{A}_{ABE}$ responds to as follows:

   - FnInit$(\cdot, MK, PP)$: run as per Algorithm 10.
   - Register$(\cdot, MK, PP)$: run as per Algorithm 11.
   - Certify$(\mathtt{mode}, (\mathbb{O}, \psi), l, \mathcal{F}_i, S_i, MK, PP)$: $\mathcal{A}_{ABE}$ will immediately return $\perp$ if the set of queried functions includes the challenge function F and the query is for a server $S_i \notin \overline{R}$ since the adversary would be given an evaluation key for a valid server at the

challenge time otherwise, leading to a trivial win. It will also return $\perp$ if the current time period t equals the challenge time period, set to be q_t and (other than S_i) there exists a server that the adversary claimed would be revoked in $\overline{R}$ but is not currently revoked (listed in Q_{Rev}). If neither failure condition occurs, then $\mathcal{A}_{ABE}$ removes S_i from the revocation list and simulates running HPVC.Certify as follows.

To generate the evaluation key for the queried parameters, $\mathcal{A}_{ABE}$ uses the KeyGen oracle in the rkDPABE game. It first updates the relevant list entries as specified. Then it sets $\psi' = \psi \cup l$ and $\mathbb{O}' = \mathbb{O} \wedge l$ and makes an oracle query to $\mathcal{C}$ for $\mathcal{O}^{\mathsf{KeyGen}}(S_i, (\mathbb{O}', \psi'), MK, PP)$ as in Oracle Query 6. $\mathcal{C}$ shall generate a rkDPABE decryption key $SK_{\psi'}$ if and only if $\tilde{\omega} \notin \mathbb{O}'$ or $\psi' \notin \tilde{\mathbb{S}}$ or $S_i \in \overline{R}$.

Now, by construction (Step 2), $\tilde{\omega} \in \mathbb{O}'$ only if the label $l^\star = l$. If the labels *do not* match (e.g. if the query is for a different function than F), then $\mathcal{C}$ may generate the key, which $\mathcal{A}_{ABE}$ will receive as SK^0_{ABE}.

If, on the other hand, the labels *do* match, then since each data label uniquely describes its input, $l^\star = l$ if and only if $(\psi^\star = \psi)$ and $(\mathbb{O}^\star = \mathbb{O})$. In the case of the challenge mode being VDC, we chose $(\tilde{\omega}, \tilde{\mathbb{S}})$ in Step 2 specifically such that these were not satisfied on $(\mathbb{O}^\star, \psi^\star)$ (and hence, by the above equality, on $(\mathbb{O}, \psi)$). Thus, $\mathcal{C}$ may generate the key SK^0_{ABE}. If the challenge mode is RPVC however, then the label l corresponds to a function identifier, and since $l^\star = l$ it corresponds to the challenge function. Now, since the KDC is assumed to check that the label corresponds to the input to Certify (by applying a injective mapping for example), it must be that the challenge function $F \in \mathcal{F}_i$ (in particular, $\mathcal{F}_i = \{F\}$). Hence, if $S_i \notin \overline{R}$, $\mathcal{A}_{ABE}$ would have returned $\perp$ immediately as specified in line 2 of Oracle Query 1. Thus, to have got to this point in the execution, it must be that $S_i \in \overline{R}$ and $\mathcal{C}$ may generate the secret key (as S_i will be revoked for the challenge, this key will not lead to a trivial win).

$\mathcal{A}_{ABE}$ also must request an update key by making a KeyUpdate oracle query to $\mathcal{C}$. $\mathcal{C}$ will return a valid key unless the current time period is $t = \tilde{t}$ and $\overline{R} \not\subseteq Q_{Rev}$. However, if the failure case were true, then $\mathcal{A}_{ABE}$ would already have returned $\perp$ by line 2 of Oracle Query 1.

If in RPVC mode, $\mathcal{A}_{ABE}$ additionally generates a key using the second system parameters (which he owns) for $(\overline{\mathbb{O}}, \overline{\psi})$.

- Revoke$(\tau_{(\mathbb{O},\psi),(\omega,\mathbb{S})}, (\mathbb{O}, \psi), (\omega, \mathbb{S}), MK, PP)$: In response to a Revoke query, $\mathcal{A}_{ABE}$ will increment the time counter and return $\perp$ if the token does not call for an entity to be revoked (note that this is consistent with the Revoke algorithm). It also returns $\perp$ if the updated time period is the challenge time q_t and Q_{Rev} (with the inclusion of the about-to-be-revoked server S_i) does not contain all the servers declared in $\overline{R}$ (as the adversary has not executed a valid sequence of oracle queries in this case). Otherwise, S_i is added to Q_{Rev} and $\mathcal{A}_{ABE}$ simulates running HPVC.Revoke as follows.

  It runs Algorithm 17 as written with the exception of line 4, where it will instead make a KeyUpdate oracle query to $\mathcal{C}$. $\mathcal{C}$ will generate a valid update key unless the time period is $\tilde{t} = q_t$ and there exists a server on $\overline{R}$ that is not currently revoked. However this is precisely the condition under which $\mathcal{A}_{ABE}$ would have returned $\perp$, so $\mathcal{C}$ can always return a valid key.

7. Once $\mathcal{A}_{HPVC}$ has finished this phase of querying (in particular, after q_t Revoke queries), $\mathcal{A}_{ABE}$ must check that all servers listed in $\overline{R}$ are indeed currently revoked. If not, then $\mathcal{A}_{HPVC}$ loses the game as he has not performed correctly.

8. To generate the challenge input for either **Game 0** or **Game 1**, $\mathcal{A}_{ABE}$ begins by choosing a random bits $RK_{(\omega^\star,\mathbb{S}^\star)} = b$ and s, and three random messages m_0, m_1 and m' from the message space.

   $\mathcal{A}_{ABE}$ sends the messages m_0 and m_1 to $\mathcal{C}$ as the challenge messages for the DP-ABE game. $\mathcal{C}$ chooses a random bit $b^\star$ and returns $CT^\star \leftarrow \mathsf{DPABE.Encrypt}(m_{b^\star}, (\tilde{\omega}, \tilde{\mathbb{S}}), \tilde{t}, MPK_{\text{ABE}})$.

   - In RPVC mode, recall that $\tilde{\omega} = \omega^\star \cup l^\star = x_i^\star \cup l(F)$ and $\tilde{\mathbb{S}} = \mathbb{S}^\star \wedge l^\star = \{\{T_S^0\}\} \wedge l(F)$. $\mathcal{A}_{ABE}$ sets c_b to be $CT^\star$ and generates $c_{1-b} \leftarrow \mathsf{DPABE.Encrypt}(m', (\tilde{\omega}, \tilde{\mathbb{S}}), \tilde{t}, MPK^1_{\text{ABE}})$ himself. It sets $VK_b = g(m_s)$ and $VK_{1-b} = g(m')$.
   - In VDC mode:
     - If $r = 1$: recall, $\tilde{\omega} = \omega^\star \cup l^\star = \{T_O^0\} \cup l(x_i^\star)$ and $\tilde{\mathbb{S}} = \overline{\mathbb{S}^\star} \wedge l^\star = \overline{F} \wedge l(x_i^\star)$. $\mathcal{A}_{ABE}$ generates $c_b \leftarrow \mathsf{DPABE.Encrypt}(m', (\omega^\star \cup l^\star, \mathbb{S}^\star \wedge l^\star), \tilde{t}, MPK^0_{\text{ABE}})$ himself, and sets c_{1-b} to be $CT^\star$. It sets $VK_b = g(m')$ and $VK_{1-b} = g(m_s)$.
     - If $r = 0$: recall, $\tilde{\omega} = \omega^\star \cup l^\star = \{T_O^0\} \cup l(x_i^\star)$ and $\tilde{\mathbb{S}} = \mathbb{S}^\star \wedge l^\star = F \wedge l(x_i^\star)$ $\mathcal{A}_{ABE}$ sets c_b to be $CT^\star$ and generates $c_{1-b} \leftarrow \mathsf{DPABE.Encrypt}(m', (\omega^\star \cup l^\star, \overline{\mathbb{S}^\star} \wedge l^\star), \tilde{t}, MPK^0_{\text{ABE}})$ himself. It sets $VK_b = g(m_s)$ and $VK_{1-b} = g(m')$.

   Finally, $\mathcal{A}_{ABE}$ sets $\sigma_{(\omega^\star,\mathbb{S}^\star)} = (c_b, c_{1-b})$, $VK_{(\omega^\star,\mathbb{S}^\star)} = (VK_b, VK_{1-b}, L_{Reg})$ and $RK_{(\omega^\star,\mathbb{S}^\star)} = b$. Note that s is essentially $\mathcal{A}_{ABE}$'s guess for the value of $b^\star$ chosen by $\mathcal{C}$.

9. $\mathcal{A}_{HPVC}$ is provided the output of $\mathsf{ProbGen}$ and again given oracle access which is handled in the same manner as in Stage 6. Eventually, it outputs $\theta_{(\mathbb{O}^\star,\psi^\star),(\omega^\star,\mathbb{S}^\star)}$ which it believes is a valid forgery (i.e. that it will be accepted yet does not correspond to the correct value of $F(x_i^\star)$).

10. $\mathcal{A}_{ABE}$ parses $\theta_{(\mathbb{O}^\star,\psi^\star),(\omega^\star,\mathbb{S}^\star)}$ as $(d_b, d_{1-b}, S_{i^\star}, \gamma)$ and using the retrieval key $RK_{(\mathbb{O}^\star,\psi^\star),(\omega^\star,\mathbb{S}^\star)} = b$, finds d_0 and d_1. One of d_0 and d_1 will be $\perp$ (by construction) and we denote the other value by Y.

    If $g(Y) = g(m_s)$, $\mathcal{A}_{ABE}$ outputs a guess $b' = s$ of $b^\star$ chosen by $\mathcal{C}$, and otherwise guesses $b' = (1 - s)$.

If $s = b^\star$ (the challenge bit chosen by $\mathcal{C}$), we observe that the above corresponds to **Game 0** (since the verification key comprises $g(m')$ where m' is the message a legitimate server could recover, and $g(m_s)$ where m_s is the other plaintext). Alternatively, $s = 1 - b^\star$ and the distribution of the above experiment is identical to **Game 1** (since the verification key comprises the legitimate message and a random message $m_{1-b^\star}$ that is unrelated to the ciphertext).

Now, we consider the advantage of this constructed $\mathcal{A}_{ABE}$ playing the IND-sHRSS game for $\mathsf{rkDPABE}$ Recall that by assumption, $\mathcal{A}_{HPVC}$ has a non-negligible advantage δ in distinguishing between **Game 0** and **Game 1** – that is

$$|\Pr(\mathbf{Exp}^0_{\mathcal{A}_{HPVC}}[\mathcal{HPVC}, F, q_t, 1^\kappa]) - \Pr(\mathbf{Exp}^1_{\mathcal{A}_{HPVC}}[\mathcal{HPVC}, F, q_t, 1^\kappa])| \geqslant \delta$$

where $\mathbf{Exp}^i_{\mathcal{A}_{HPVC}}[\mathcal{HPVC}, F, q_t, 1^\kappa]$ denotes the output of running $\mathcal{A}_{HPVC}$ in **Game** i.

$$
\begin{aligned}
\Pr('= b^\star) &= \Pr(s = b^\star)\Pr(b' = b^\star | s = b^\star) + \Pr(s \neq b^\star)\Pr(b' = b^\star | s \neq b^\star) \\
&= \frac{1}{2}\Pr(g(Y) = g(m_s) | s = b^\star) + \frac{1}{2}\Pr(g(Y) \neq g(m_s) | s \neq b^\star) \\
&= \frac{1}{2}\Pr(\mathbf{Exp}^0_{\mathcal{A}_{HPVC}}[\mathcal{HPVC}, F, q_t, 1^\kappa]) + \frac{1}{2}\left(1 - \Pr(g(\sigma_{y^\star}) = g(m_s) | s \neq b^\star)\right) \\
&= \frac{1}{2}\Pr(\mathbf{Exp}^0_{\mathcal{A}_{HPVC}}[\mathcal{HPVC}, F, q_t, 1^\kappa]) + \frac{1}{2}\left(1 - \Pr(\mathbf{Exp}^1_{\mathcal{A}_{HPVC}}[\mathcal{HPVC}, F, q_t, 1^\kappa])\right) \\
&= \frac{1}{2}\left(\Pr(\mathbf{Exp}^0_{\mathcal{A}_{HPVC}}[\mathcal{HPVC}, F, q_t, 1^\kappa]) - \Pr(\mathbf{Exp}^1_{\mathcal{A}_{HPVC}}[\mathcal{HPVC}, F, q_t, 1^\kappa]) + 1\right) \\
&\geqslant \frac{1}{2}(\delta + 1)
\end{aligned}
$$

Hence,

$$
\begin{aligned}
Adv_{\mathcal{A}_{ABE}} &\geqslant \left|\Pr(b^\star = b') - \frac{1}{2}\right| \\
&\geqslant \left|\frac{1}{2}(\delta + 1) - \frac{1}{2}\right| \\
&\geqslant \frac{\delta}{2}
\end{aligned}
$$

Hence, if $\mathcal{A}_{HPVC}$ has advantage δ at distinguishing these games then $\mathcal{A}_{ABE}$ can win the IND-sHRSS game for rkDPABE with non-negligible probability. Thus since we assumed the rkDPABE scheme to be secure, we conclude that $\mathcal{A}_{HPVC}$ cannot distinguish **Game 0** from **Game 1** with non-negligible probability.

Game 1 to Game 2. The transition from **Game 1** to **Game 2** is simply to set the value of m' to no longer be random but instead to correspond to the challenge w in the one-way function inversion game. We argue that the adversary has no distinguishing advantage between these games since the new value is independent of anything else in the system bar the verification key $g(w)$ and hence looks random to an adversary with no additional information (in particular, $\mathcal{A}_{HPVC}$ does not see the challenge for the one-way function as this is played between $\mathcal{C}$ and $\mathcal{A}_{ABE}$).

Final Proof We now show that using $\mathcal{A}_{HPVC}$ in **Game 2**, $\mathcal{A}_{ABE}$ can invert the one-way function g – that is, given a challenge $z = g(w)$ we can recover w. Specifically, during ProbGen, we choose the messages as follows:

- if $r = 1$, we implicitly set m_{1-b} to be w and the corresponding verification key component to be z. We choose m_b and the other verification key component randomly as usual.
- if $r = 0$, we implicitly set m_b to be w and set the corresponding verification key component to be z. We choose m_{1-b} and the other verification key component randomly as usual.

Now, since $\mathcal{A}_{HPVC}$ is assumed to be successful, it will output a forgery comprising the plaintext encrypted under the unsatisfied function (F or $\overline{F}$). By construction, this will be w (and the adversary's view in consistent since the verification key is simulated correctly using z). $\mathcal{A}_{ABE}$ can therefore forward this result to $\mathcal{C}$ in order to invert the one-way function with the same non-negligible probability that $\mathcal{A}_{HPVC}$ has against the selective, semi-static Public Verifiability game.

We conclude that if the rkDPABE scheme is IND-sHRSS secure and the one-way function is hard-to-invert, then the $HPVC$ as defined by Algorithms 9–17 is secure in the sense of selective, semi-static Public Verifiability.

□

3.5.2 Proof of Revocation

Lemma 4. *The HPVC construction defined by Algorithms 9–17 is secure in the sense of selective, semi-static Revocation (Game 8) under the same assumptions as in Theorem 2.*

Proof. The proof follows in a very similar way as the proof for Lemma 3.
Let $\mathcal{A}_{HPVC}$ be an adversary with non-negligible advantage against the selective, semi-static Revocation game (Game 8) when instantiated with Algorithms 9–16. We define the following three games:

- **Game 0.** This is the correct selective, semi-static Revocation game as in Game 8.
- **Game 1.** This differs from **Game 0** in that ProbGen no longer returns an encryption of m_0 and m_1. Instead, a random message $m' \neq m_0, m_1$ is chosen and we replace the plaintext associated with the *unsatisfied* function by m'.
- **Game 2.** This is the same as **Game 1** except that m' is implicitly chosen to be the challenge input w in the one-way function game.

By hopping from **Game 0** to **Game 2**, we can show that an adversary with non-negligible advantage against selective, semi-static Revocation can be used to construct an adversary that inverts the one-way function g. The transition to **Game 2** is identical to that given in the proof of Lemma 3, and we do not replicate it here.

We now show that using $\mathcal{A}_{HPVC}$ in **Game 2** as a subroutine, $\mathcal{A}_{ABE}$ can invert the one-way function g – that is, given a challenge $z = g(w)$ $\mathcal{A}_{ABE}$ can recover w. To do so, during ProbGen, $\mathcal{A}_{ABE}$ tosses a random coin $v \xleftarrow{\$} \{0, 1\}$ and chooses the messages as follows. It implicitly sets $m_v = w$ and sets the corresponding verification key component to be z. The remaining message m_{1-v} is chosen randomly and the remainder of the verification key computed as usual.

Now, since $\mathcal{A}_{HPVC}$ is assumed to be successful, it will output a valid forgery for one of these messages. If $\mathcal{A}_{ABE}$ has guess v correctly (which he does with probability $\frac{1}{2}$), this will be w (and the adversary's view is consistent since the verification key is simulated correctly using z). $\mathcal{A}_{ABE}$ can therefore forward this result to $\mathcal{C}$ in order to invert the one-way function. Thus if $\mathcal{A}_{VC}$ has non-negligible advantage ϵ against the selective, semi-static Revocation game, then $\mathcal{A}_{ABE}$ wins with the non-negligible probability $\frac{\epsilon}{2}$.

We conclude that if the rkDPABE scheme is IND-sHRSS secure and the one-way function is hard-to-invert, then the $HPVC$ as defined by Algorithms 9–17 is secure in the sense of selective, semi-static Revocation.

□

4 Enforcing Access Control in VC with DP-ABE

While the introduction of HPVC built on DP-ABE allows for a single system that supports PVC and VDC functionality, both options use only a single-policy mode. We can also use the full power of DP-ABE to enforce restrictions on how entities may behave within the system. The notion of PVC-AC enforces access control in the PVC setting such that only servers that meet an authorization policy attached to the encoded input may produce valid results (and hence be rewarded for their effort).

Alderman et al. [3] introduced PVC-AC with the motivation that, in practice, it is likely that servers be selected from a (large) pool of available servers based on a system-dependent mechanism (e.g. resource availability or a bidding process). (This contrasts with prior models [29] where a client chose a server up-front with which to set up a PVC system.) Thus delegators have less knowledge about the selected server and may not authenticate them beforehand. The PVC-AC construction of [3] used symmetric encryption and Key Assignment Schemes such that only authorized entities could derive decryption keys. However, delegators and verifiers must be registered by the KDC. This is partly due to the policies being enforced (e.g. such that delegators may outsource only certain computations) but also due to the use of symmetric primitives – to encrypt an input such that only authorized servers may decrypt, delegators must be privately issued the symmetric key. Thus, the scheme is not strictly *publicly delegable* – any party may register to be a delegator but delegation does not depend *only* on public information, and similarly for verification.

Here, we give a more relaxed framework that retains public verifiability and public delegability[6]. In some sense, servers are already authorized for functions by virtue of being issued evaluation keys. However, we believe not all outsourced computations should be considered purely in terms of functions and that access control policies in this setting should allow for additional context. For example, a government contractor that subscribes to a verifiable software-as-a-service system may, due to the nature of its work, require that servers be physically located within the same country. Alternatively, input data may affect access control requirements – for example, an averaging function over rainfall data may be fairly innocuous but when applied to military spending may be more sensitive.

DP-ABE allows two policies to be simultaneously enforced: an objective (key-policy) $\mathbb{O}$ and a subjective (ciphertext-policy) $\mathbb{S}$. Informally, for PVC-AC, we use the objective policy to evaluate an outsourced computation (as before) whilst the subjective policy will additionally be used to enforce access control on the server. Let $\mathcal{U}_C$ be the attribute universe used for PVC in Section 3 and let $\mathcal{U}_A$ be a universe of authorization attributes (disjoint from $\mathcal{U}_C$). Servers are assigned both an evaluation key for a function F formed over U_C *and* a set of descriptive attributes $s \subseteq \mathcal{U}_A$ describing their authorization rights. $\mathsf{ProbGen}$ operates on both the input data x *and* an authorization policy P which dictates the sets of necessary authorization attributes to perform this computation. Decryption, and hence computation, may occur if and only if $F(x)$ outputs 1 *and* the server satisfies the authorization policy – $s \in P$. If either property fails, no information about the encrypted message is learnt. For example, s may be $\{\mathtt{UK}, \mathtt{Capacity} = \mathtt{3TB}\}$ while $P = (\mathtt{UK}) \vee ((\mathtt{clearance} = \mathtt{Secret}) \wedge (\mathtt{USA})) = \{\{\mathtt{UK}\},\{\mathtt{clearance} = \mathtt{Secret}, \mathtt{USA}\}\}$. Similarly, we could, in the VDC setting allow the subjective policy to specify the function F whilst the objective policy contains the authorization policy P. Then we can restrict the delegators that may request a computation to only those that can provide certain attributes, but we do not discuss this in detail here. Table 2 reiterates the choice of parameters from Table 1 along with appropriate choices for access control functionality. The instantiation for PVC-AC is identical to that given in Algorithms 9–17 with these choices of parameters – where dummy attributes and policies were used before, we substitute for the authorization attributes s and policy P respectively, and we get this additional functionality for free. We give additional details and security models in Appendix C.

[6] At present we do not consider input privacy against unauthorized servers, but note that a dual-policy predicate encryption scheme could be used. Since the construction and techniques make black-box use of the DP-ABE scheme, this exchange can easily be made if a suitable dual-policy predicate encryption scheme is found.

Table 2: Parameter definitions for different modes

mode	$\mathbb{O}$	ψ	$l(\mathbb{O}, \psi)$	ω	$\mathbb{S}$	$l(\omega, \mathbb{S})$	$\mathcal{F}_i$
PVC	F	$\{T_S\}$	$l(F)$	x	$\{\{T_S\}\}$	$l(F)$	$\{F\}$
VDC	$\{\{T_O\}\}$	x	$l(x)$	$\{T_O\}$	F	$l(x)$	$\{F_1, \ldots, F_m\}$
PVC-AC	F	s	$l(F)$	x	P	$l(F)$	$\{F\}$

5 Conclusion

We have introduced the notion of Hybrid Publicly Verifiable Computation that flexibly supports multiple modes of operation to meet the diverse user requirements of a large multi-user system. We capture the notions of RPVC, RPVC with access control on servers (maintaining public delegability and verification), and a reversed model, VDC, where servers own static data. We have seen that this notion leads to a natural and novel use of DP-ABE. In future work, we will consider the use of multiple KDCs in a DP-ABE scheme such that, in HPVC, the responsibilities for assigning function evaluation keys and for assigning security attributes are not borne by a single entity. We believe that in practice, it is more likely that entities will be authoritative on only one of these areas (and that the security KDC could also be used in other systems). This amounts to splitting the KeyGen operations for the KP and CP parts of the DP-ABE scheme, yet ensuring that the scheme remains secure by tying these keys together by using a global identifier [16, 26].We will also further investigate our revocable DP-ABE scheme to compare the efficiency of revoking between the key- and ciphertext-policies. Finally, interesting open questions are whether predicate encryption could enable private information retrieval, and whether multi-authority techniques could allow servers themselves to generate evaluation keys for *only* their own data. Furthermore we will investigate techniques to allow dynamically updatable data stored on the server.

References

[1] J. Alderman and J. Crampton. On the use of key assignment schemes in authentication protocols. In J. Lopez, X. Huang, and R. Sandhu, editors, *Network and System Security*, volume 7873 of *Lecture Notes in Computer Science*, pages 607–613. Springer Berlin Heidelberg, 2013.

[2] J. Alderman, C. Janson, C. Cid, and J. Crampton. Revocation in publicly verifiable outsourced computation. In D. Lin, M. Yung, and J. Zhou, editors, *Information Security and Cryptology - 10th International Conference, Inscrypt 2014, Beijing, China, December 13-15, 2014, Revised Selected Papers*, volume 8957 of *Lecture Notes in Computer Science*. Springer, 2014.

[3] J. Alderman, C. Janson, C. Cid, and J. Crampton. Access control in publicly verifiable outsourced computation. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, pages 657–662, New York, NY, USA, 2015. ACM.

[4] D. Apon, J. Katz, E. Shi, and A. Thiruvengadam. Verifiable oblivious storage. In H. Krawczyk, editor, *Public-Key Cryptography - PKC 2014 - 17th International Conference*

on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings, volume 8383 of *Lecture Notes in Computer Science*, pages 131–148. Springer, 2014.

[5] N. Attrapadung and H. Imai. Attribute-based encryption supporting direct/indirect revocation modes. In M. G. Parker, editor, *IMA Int. Conf.*, volume 5921 of *Lecture Notes in Computer Science*, pages 278–300. Springer, 2009.

[6] N. Attrapadung and H. Imai. Dual-policy attribute based encryption. In M. Abdalla, D. Pointcheval, P.-A. Fouque, and D. Vergnaud, editors, *ACNS*, volume 5536 of *Lecture Notes in Computer Science*, pages 168–185, 2009.

[7] N. Attrapadung and H. Imai. Dual-policy attribute based encryption: Simultaneous access control with ciphertext and key policies. *IEICE Transactions*, 93-A(1):116–125, 2010.

[8] M. Backes, M. Barbosa, D. Fiore, and R. M. Reischuk. Adsnark: Nearly practical and privacy-preserving proofs on authenticated data. Cryptology ePrint Archive, Report 2014/617, 2014. `http://eprint.iacr.org/`.

[9] M. Backes, D. Fiore, and R. M. Reischuk. Verifiable delegation of computation on outsourced data. In A. Sadeghi, V. D. Gligor, and M. Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 863–874. ACM, 2013.

[10] E. Ben-Sasson, A. Chiesa, D. Genkin, and E. Tromer. Fast reductions from rams to delegatable succinct constraint satisfaction problems: extended abstract. In R. D. Kleinberg, editor, *Innovations in Theoretical Computer Science, ITCS '13, Berkeley, CA, USA, January 9-12, 2013*, pages 401–414. ACM, 2013.

[11] S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In Rogaway [30], pages 111–131.

[12] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334. IEEE Computer Society, 2007.

[13] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In S. Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 326–349. ACM, 2012.

[14] D. Boneh, C. Gentry, and B. Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In V. Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 258–275. Springer, 2005.

[15] D. Catalano, D. Fiore, R. Gennaro, and K. Vamvourellis. Algebraic (trapdoor) one-way functions and their applications. In Sahai [31], pages 680–699.

[16] M. Chase. Multi-authority attribute based encryption. In S. P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 515–534. Springer, 2007.

[17] S. G. Choi, J. Katz, R. Kumaresan, and C. Cid. Multi-client non-interactive verifiable computation. In Sahai [31], pages 499–518.

[18] K. Chung, Y. T. Kalai, F. Liu, and R. Raz. Memory delegation. In Rogaway [30], pages 151–168.

[19] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[20] D. Fiore and R. Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In T. Yu, G. Danezis, and V. D. Gligor, editors, *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 501–512. ACM, 2012.

[21] A. Fujii, G. Ohtake, G. Hanaoka, and K. Ogawa. Anonymous authentication scheme for subscription services. In B. Apolloni, R. J. Howlett, and L. C. Jain, editors, *KES (3)*, volume 4694 of *Lecture Notes in Computer Science*, pages 975–983. Springer, 2007.

[22] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In T. Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2010.

[23] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct nizks without pcps. In T. Johansson and P. Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645. Springer, 2013.

[24] S. Goldwasser, V. Goyal, A. Jain, and A. Sahai. Multi-input functional encryption. Cryptology ePrint Archive, Report 2013/727, 2013. `http://eprint.iacr.org/`.

[25] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In A. Juels, R. N. Wright, and S. D. C. di Vimercati, editors, *ACM Conference on Computer and Communications Security*, pages 89–98. ACM, 2006.

[26] A. B. Lewko and B. Waters. Decentralizing attribute-based encryption. In K. G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 568–588. Springer, 2011.

[27] K. Ohta, T. Okamoto, and K. Koyama. Membership authentication for hierarchical multigroups using the extended Fiat-Shamir scheme. In I. Damgård, editor, *EUROCRYPT*, volume 473 of *Lecture Notes in Computer Science*, pages 446–457. Springer, 1990.

[28] C. Papamanthou, E. Shi, and R. Tamassia. Signatures of correct computation. In Sahai [31], pages 222–242.

[29] B. Parno, M. Raykova, and V. Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In R. Cramer, editor, *TCC*, volume 7194 of *Lecture Notes in Computer Science*, pages 422–439. Springer, 2012.

[30] P. Rogaway, editor. *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*. Springer, 2011.

[31] A. Sahai, editor. *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, volume 7785 of *Lecture Notes in Computer Science*. Springer, 2013.

[32] S. E. Schechter, T. Parnell, and A. J. Hartemink. Anonymous authentication of membership in dynamic groups. In M. K. Franklin, editor, *Financial Cryptography*, volume 1648 of *Lecture Notes in Computer Science*, pages 184–195. Springer, 1999.

[33] J. Shi, J. Lai, Y. Li, R. H. Deng, and J. Weng. Authorized keyword search on encrypted data. In M. Kutylowski and J. Vaidya, editors, *Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part I*, volume 8712 of *Lecture Notes in Computer Science*, pages 419–435. Springer, 2014.

[34] J. van den Hooff, M. F. Kaashoek, and N. Zeldovich. Versum: Verifiable computations over large public logs. In G. Ahn, M. Yung, and N. Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 1304–1316. ACM, 2014.

[35] B. Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi, editors, *Public Key Cryptography - PKC 2011 - 14th International Conference on Practice and Theory in Public Key Cryptography, Taormina, Italy, March 6-9, 2011. Proceedings*, volume 6571 of *Lecture Notes in Computer Science*, pages 53–70. Springer, 2011.

[36] T. White. *Hadoop: the definitive guide: the definitive guide.* " O'Reilly Media, Inc.", 2009.

[37] L. F. Zhang and R. Safavi-Naini. Private outsourcing of polynomial evaluation and matrix multiplication using multilinear maps. In M. Abdalla, C. Nita-Rotaru, and R. Dahab, editors, *Cryptology and Network Security - 12th International Conference, CANS 2013, Paraty, Brazil, November 20-22. 2013. Proceedings*, volume 8257 of *Lecture Notes in Computer Science*, pages 329–348. Springer, 2013.

A Background and Related Work

A.1 Verifiable Outsourced Computation

The concept of non-interactive verifiable computation was introduced by Gennaro et al. [22] and may be seen as a protocol between two polynomial-time parties: a *client*, C, and a *server*, S. A successful run of the protocol results in the provably correct computation of $F(x)$ by the server for an input x supplied by the client. More specifically, a VC scheme comprises the following steps [22]:

1. C computes evaluation information EK_F that is given to S to enable it to compute F (pre-processing)
2. C sends the encoded input σ_x to S (input preparation)
3. S computes $y = F(x)$ using EK_F and σ_x and returns an encoding of the output $\theta_{F(x)}$ to C (output computation)
4. C checks whether $\theta_{F(x)}$ encodes $F(x)$ (verification)

KeyGen may be computationally expensive (and amortized over multiple computations of F) but the remaining operations should be efficient for the client.

Parno et al. [29] introduced *Publicly Verifiable Computation* (PVC), which we discuss further in Appendix A.2. Some works have also considered the multi-client case in which the input data to be sent to the server is shared between multiple clients, and notions such as input privacy

Table 3: PVC using KP-ABE

Abstract PVC parameter	**Parameter in KP-ABE instantiation**
EK_F	$SK_{\mathbb{A}_F}$
PK_F	Master public key PP
σ_x	Encryption of m using PP and A_x
σ_y	m or $\perp$
$VK_{F,x}$	$g(m)$

become more important. Choi et al. [17] extended the garbled circuit approach [22] using a proxy-oblivious transfer primitive to achieve input privacy in a non-interactive scheme. Recent work of Goldwasser et al. [24] extended the construction of Parno et al. [29] to allow multiple clients to provide input to a functional encryption algorithm.

Other works [11, 20, 15] design VC schemes for specific functions (e.g. matrix multiplication) without using FHE and even partially achieve function privacy. In particular [37] uses multilinear maps to achieve the aforementioned.

A.2 PVC using Key-Policy Attribute-based Encryption.

Parno et al. [29] provide a concrete instantiation of PVC using KP-ABE[7] for the case when F is a Boolean function [29]. Define a universe $\mathcal{U}$ of n attributes and associate $V \subseteq \mathcal{U}$ with a binary n-tuple in which the ith place is 1 if and only if the ith attribute is in V. We call this the *characteristic tuple* of V. Thus, there is a natural one-to-one correspondence between n-tuples and attribute sets; we write A_x to denote the set associated with x. An alternative way to view this is to let $\mathcal{U} = \{A_1, A_2, \ldots, A_n\}$. Then, a bit string $\bar{v}$ of length n is the characteristic tuple of the set $V \subseteq \mathcal{U}$ if $V = \{A_i : \overline{v_i} = 1\}$. A function $F : \{0,1\}^n \to \{0,1\}$ is monotonic if $x \leqslant y$ implies $F(x) \leqslant F(y)$, where $x = (x_1, \ldots, x_n)$ is less than or equal to $y = (y_1, \ldots, y_n)$ if and only if $x_i \leqslant y_i$ for all i. For a monotonic function $F : \{0,1\}^n \to \{0,1\}$, the set $\{x \in \{0,1\}^n : F(x) = 1\}$ defines a monotonic access structure which we denote $\mathbb{A}_F$. The mapping between PVC and KP-ABE parameters is shown in Table 3. Informally, for a Boolean function F, the client generates a private key $SK_{\mathbb{A}_F}$ using the KeyGen algorithm. Given an input x, a client encrypts a random message m "with" A_x using the $\mathsf{Encrypt}$ algorithm and publishes $VK_{F,x} = g(m)$ where g is a suitable one-way function (e.g. a pre-image resistant hash function). The server decrypts the message using the $\mathsf{Decrypt}$ algorithm, which will either return m (when $F(x) = 1$) or $\perp$. The server returns m to the client. Any client can test whether the value returned by the server is equal to $g(m)$. Note, however, that a "rational" malicious server will always return $\perp$, since returning any other value will (with high probability) result in the verification algorithm returning a reject decision. Thus, it is necessary to have the server compute both F and its "complement" (and for both outputs to be verified). The interested reader may also consult the original paper for further details [29]. Note that, to compute the private key $SK_{\mathbb{A}_F}$, it is necessary to identify all minimal elements x of $\{0,1\}^n$ such that $F(x) = 1$. There may be exponentially many such x. Thus, the initial phase is indeed computationally expensive for the client.

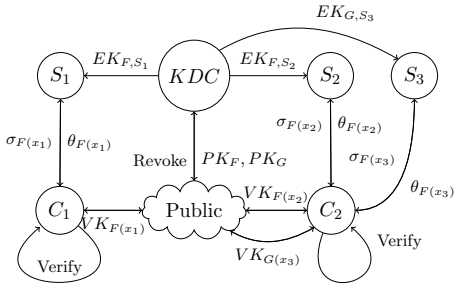

Figure 2: The operation of Revocable PVC

A.3 Revocable PVC

A revocable PVC scheme [2] comprises the algorithms Setup, FnInit, Register, Certify, ProbGen, Compute, BVerif, Retrieve and Revoke which correspond to the following steps, and as in Figure 2 :

- The KDC generates public parameters, issues personalised secret keys, and evaluation keys to servers and publishes function delegation information.
- To outsource the evaluation of $F(x)$, a delegator C, sends an encoded input $\sigma_{F(x)}$ to a server S, and publishes verification tokens for the computation.
- S uses $\sigma_{F(x)}$ and an evaluation key for F to produce an encoded output $\theta_{F(x)}$ (sent to C, the manager, or published depending on the system architecture).
- Any entity can use the verification token to blind verify correctness of the output. The verifier may not learn the value of $F(x)$ if not in possession of the retrieval key. If S cheated they may report S to the KDC for revocation.
- If blind verification was successful, a party possessing the retrieval key $RK_{F(x)}$ can recover $F(x)$.
- The KDC may revoke a cheating server to prevent it computing F in the future (and hence from receiving any reward for future work).

We now give an updated definition for Revocable PVC to be in line with the notation used in our Hybrid PVC definition in Section 3:

Definition 11. *A* Revocable PVC scheme *comprises the following algorithms:*

1. $(PP, MK) \leftarrow \mathsf{Setup}(1^\kappa)$
2. $PK_F \leftarrow \mathsf{FnInit}(F, MK, PP)$
3. $SK_{S_i} \leftarrow \mathsf{Register}(S_i, MK, PP)$
4. $EK_{F,S_i} \leftarrow \mathsf{Certify}(S_i, F, l(F), MK, PP)$
5. $(\sigma_{F(x)}, VK_{F(x)}, RK_{F(x)}) \leftarrow \mathsf{ProbGen}(x, l(F), PK_F, PP)$
6. $\theta_{F(x)} \leftarrow \mathsf{Compute}(\sigma_{F(x)}, EK_x, SK_{S_i}, PP)$
7. $y_{F(x)} \leftarrow \mathsf{Verify}(\theta_{F(x)}, RT_{F(x)}, VK_{F(x)}, RK_{F(x)}PP)$: Verification comprises two steps.

[7] If input privacy is required then a predicate encryption scheme could be used in place of the KP-ABE scheme.

- $(RT_{F(x)}, \tau_{\theta_{F(x)}}) \leftarrow \mathsf{BVerif}(\theta_{F(x)}, VK_{F(x)}, PP)$
- $y_{F(x)} \leftarrow \mathsf{Retrieve}(RT_{F(x)}, VK_{F(x)}, RK_{F(x)} PP)$

8. $\{EK_{F,S'}\}$ or $\perp \leftarrow \mathsf{Revoke}(\tau_{\theta_{F(x)}}, MK, PP)$

A.4 Dual-Policy Attribute-Based Encryption

Dual-Policy Attribute-Based Encryption (DP-ABE) [7] conjunctively combines the two standard approaches of Key-Policy ABE [25] (supporting objective policies) and Ciphertext-Policy ABE [12] (enforcing subjective policies) such that both the ciphertext and the decryption key comprise an attribute set *and* an access policy. Thus, the ciphertext is associated with both an subjective access policy (as per CP-ABE) detailing which entities may decrypt *and* an objective attribute set describing the data. Decryption keys comprise an objective access policy (as per KP-ABE) and a subjective attribute set. Decryption succeeds if and only if *both* attribute sets satisfy their corresponding access policies. More formally, a DP-ABE scheme is defined as follows (definitions for KP-ABE and CP-ABE can be found by ignoring the relevant policies and attribute sets):

Definition 12 (Dual-Policy Attribute-based Encryption [7])**.** *A Dual-Policy Attribute-based Encryption scheme comprises four algorithms as follows:*

- $(PK, MK) \leftarrow \mathsf{Setup}(1^\kappa)$*: The randomized setup algorithm takes the security parameter as input and generates a public key PK for the system which is published, and a master secret key MK which is kept by the executor of this algorithm.*
- $CT_{\omega,\mathbb{S}} \leftarrow \mathsf{Encrypt}(m, (\omega, \mathbb{S}), PK)$*: This randomized algorithm takes in the public key, a message to be encrypted, a subjective access policy $\mathbb{S}$ and an objective attribute set ω. It outputs a ciphertext CT.*
- $\mathrm{SK}_{\mathbb{O},\psi} \leftarrow \mathsf{KeyGen}((\mathbb{O}, \psi), MK, PK)$*: This randomized algorithm takes as input the public key and master secret key output by* Setup*, as well as an objective access policy $\mathbb{O}$ and a subjective attribute set ψ. It outputs a secret decryption key $\mathrm{SK}_{\mathbb{O},\psi}$.*
- *m or $\perp \leftarrow$* $\mathsf{Decrypt}(\mathrm{SK}_{\mathbb{O},\psi}, CT_{\omega,\mathbb{S}}, PK)$*: The decrypt algorithm takes as input the public key, the secret key and the ciphertext. It outputs the correct plaintext m if and only if the set of objective attributes ω satisfies the objective access structure $\mathbb{O}$* and *the set of subjective attributes ψ satisfies the subjective policy $\mathbb{S}$ – that is, $\omega \in \mathbb{O}$ and $\psi \in \mathbb{S}$. We assume throughout that the policies and attributes are implicit from the relevant keys and ciphertexts (otherwise these can also be given as arguments to this function).*

Correctness of the decryption is defined to be: if $\mathsf{Setup}(1^\kappa) \rightarrow (PK, MK)$ then

$$\mathsf{Decrypt}\left(\mathsf{KeyGen}((\mathbb{O}, \psi), MK, PK), \mathsf{Encrypt}(m, (\omega, \mathbb{S}), PK), PK\right) \rightarrow m$$

for all m in the message space and for all $\omega \in \mathbb{O}$ and $\psi \in \mathbb{S}$. Selective security for DP-ABE is defined in Game 9.

The advantage of an adversary in the selective security game (Game 9) is defined as $\Pr[b = b'] - \frac{1}{2}$. Attrapadung et al. [6] note that the game can be changed such that we are able to handle chosen-ciphertext attacks by providing the adversary with an additional decryption oracle in the query phases.

Definition 13. *A DP-ABE scheme is secure in the selective-set security notion if all PPT adversaries have at most a negligible advantage in Game 9.*

Game 9 $\mathbf{Exp}_{\mathcal{A}}^{s\text{IND-CPA}}[\mathcal{DPABE}, 1^{\kappa}]$:

$(\omega^{\star}, \mathbb{S}^{\star}) \leftarrow \mathcal{A}(1^{\kappa})$
$(PK, MK) \leftarrow \mathsf{Setup}(1^{\kappa})$
$(m_0, m_1) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}((\cdot,\cdot),MK,PK)}(PK)$, where $|m_0| = |m_1|$
$b \xleftarrow{\$} \{0,1\}$
$CT^{\star} \leftarrow \mathsf{Encrypt}(m_b, (\omega^{\star}, \mathbb{S}^{\star}), PK)$
$b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}((\cdot,\cdot),MK,PK)}(CT^{\star}, PK)$
if $b' = b$ **then**
  return 1
else
  return 0

Oracle Query 5 $\mathcal{O}^{\mathsf{KeyGen}}((\mathbb{O}, \psi), MK, PK)$:

if $\omega^{\star} \notin \mathbb{O}$ or $\psi \notin \mathbb{S}^{\star}$ **then**
  return $SK_{\mathbb{O},\psi}$
else
  return $\perp$

A.5 MapReduce

MapReduce is a programming model for the parallel processing of large datasets using a cluster or grid of computers (nodes) which can take advantage of the locality of data to decrease transmission costs. It comprises two stages:

- **Map** A master node (or manager) takes the input and divides it into smaller sub-problems which are distributed to the worker nodes. The worker computes the function associated with the sub-problem and passes the result back to the manager.
- **Reduce** The manager collects the answer to all sub-problems and combines them to form the output to the original problem.

The functions that can be computed using a MapReduce procedure is limited to those that can be split into completely independent sub-problems (since the worker nodes cannot interact). Thus, this setting can be considered as a type of verifiable outsourced computation where each worker client is provided with the evaluation key for a sub-problem possibly unique to them and they must return a valid result from the evaluation of this problem. The data in this MapReduce setting could be either stored locally at the worker nodes or distributed by the manager. Also note that there could potentially be multiple managers who perform the reduction phase (assuming that all outputs are collected at the same time or that the function is associative so that results can be accumulated).

B Revocation in DP-ABE

In Appendix A.4 we have reviewed the notation and properties of *Dual-policy ABE*, introduced by Attrapadung et al. [7], that conjunctively combines KP-ABE and CP-ABE such that both the decryption key *and* the ciphertext comprise an attribute set and an access structure. Attrapadung et al. [5] introduced the formal notion of revocation in ABE schemes supporting two different modes: *direct revocation* and *indirect revocation*. Direct revocation allows users to specify a revocation list at the point of encryption such that periodic re-keying is not required but encryptors must have knowledge of the current revocation list. In contrast, indirect revocation requires a time period to be specified at the point of encryption and an authority to issue

updated key material at each time period to enable non-revoked entities to update their key to be functional during that time period. Alderman et al. [2] use indirectly revocable KP-ABE in order to provide a revocation mechanism in Publicly Verifiable Outsourced Computation.

B.1 Background

In this section we provide the necessary background about the notions we need in order to define a *revocable DP-ABE* scheme.

Access Structures and Linear Secret Sharing

Here we define the notion of access structures and linear secret sharing schemes recapped from [35]. These are fundamental building blocks for attribute-based encryption schemes.

Definition 14 (Access Structure)**.** *Let $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ be a set of parties (attributes). A collection $\mathbb{A} \subseteq 2^{\mathcal{P}}$ is monotone if for all B, C we have that if $B \in \mathbb{A}$ and $B \subseteq C$ then $C \in \mathbb{A}$. An access structure (resp., monotonic access structure) is a collection (resp., monotone collection) $\mathbb{A} \subseteq 2^{\mathcal{P}} \setminus \{\emptyset\}$.*

Definition 15 (Linear Secret Sharing Schemes (LSSS))**.** *Let $\mathcal{P}$ be a set of parties. Let M be a matrix of size $l \times k$. Let $\pi \colon \{1, \ldots, l\} \to \mathcal{P}$ be a function that maps a row to a party for labeling. A secret sharing scheme Π for access structure $\mathbb{A}$ over a set of parties $\mathcal{P}$ is a linear secret-sharing scheme in $\mathbb{Z}_p$ and is represented by (M, π) if it consists of two polynomial-time algorithms:*

- $\text{Share}_{(M,\pi)}$*: The algorithm takes as input $s \in \mathbb{Z}_p$ which is to be shared. It randomly chooses $y_2, \ldots, y_k \in \mathbb{Z}_p$ and let $\boldsymbol{v} = (s, y_2, \ldots, y_k)$. It outputs $M\boldsymbol{v}$ as a vector of l shares. The share $\lambda_{\pi(i)} := \boldsymbol{M_i} \cdot \boldsymbol{v}$ belongs to party $\pi(i)$, where we denote $\boldsymbol{M_i}$ as the ith row in M.*
- $\text{Recon}_{(M,\pi)}$*: The algorithm takes as input $S \in \mathbb{A}$. Let $I = \{i : \pi(i) \in S\}$. It outputs reconstruction constants $\{(i, \mu_i)\}_{i \in I}$ which is a linear reconstruction property: $\sum_{i \in I} \mu_i \cdot \lambda_{\pi(i)} = s$.*

Proposition 1. *Let (M, π) be a LSSS for access structure $\mathbb{A}$ over a set of parties $\mathcal{P}$, where M is a matrix of size $l \times k$. For all $S \notin \mathbb{A}$, there exists a polynomial time algorithm that outputs a vector $\boldsymbol{w} = (w_1, \ldots, w_k) \in \mathbb{Z}_p^k$ such that $w_1 = -1$ and for all $x \in S$ it holds that $\boldsymbol{M_i} \cdot \boldsymbol{w} = 0$.*

Bilinear Maps and Hardness Assumptions

Here we review the notions of bilinear maps and the hardness assumptions. We follow the formlization in [5, 6].

Bilinear Maps. Let $\mathbb{G}, \mathbb{G}_T$ be multiplicative groups of order p. Let g be a generator of $\mathbb{G}$. A bilinear map is a map $e \colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ for which the following hold:

1. e is bilinear: for all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}$ we have $e(u^a, v^b) = e(u, v)^{ab}$
2. e is non-degenerate: $e(g, g) \neq 1$

We say that $\mathbb{G}$ is a biliear group if the group action in $\mathbb{G}$ can be computed efficiently and there exists $\mathbb{G}_T$ for which the bilinear map $e \colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is efficiently computable.

Decision BDHE Assumption. Let $\mathbb{G}$ be a bilinear group of prime order p. The Decision q-BDHE (Bilinear Diffie-Hellman Exponent) problem [14] in $\mathbb{G}$ is stated as follows. Given a vector

$$\left(g, h, g^a, g^{(a^2)}, \ldots, g^{(a^q)}, g^{(a^{q+2})}, \ldots, g^{(a^{2q})}, Z\right) \in \mathbb{G}^{2q+1} \times \mathbb{G}_T$$

as input, determine $Z = e(g,h)^{a^{q+1}}$. We denote $g_i = g^{a^i} \in \mathbb{G}$ for shorthand. Let $\boldsymbol{y}_{g,a,q} = (g_1, \ldots, g_q, G_{q+2}, g_{2q})$. An algorithm $\mathcal{A}$ that outputs $b \in \{0,1\}$ has advantage ϵ in solving the Decision q-BDHE problem in $\mathbb{G}$ if

$$|\Pr[\mathcal{A}\left(g, h, \boldsymbol{y}_{g,a,q}, e(g_{q+1}, h)\right) = 0] - \Pr[\mathcal{A}(g, h, \boldsymbol{y}_{g,a,q}, g^s, Z) = 0]| \geq \epsilon,$$

where the probability is over the random choice of generators $g, h \in \mathbb{G}$, the random choice of $a \in \mathbb{Z}_p$, the random choice of $Z \in \mathbb{G}_T$, and the randomness of $\mathcal{A}$. We refer to the distribution on the left as $\mathcal{P}_{BDHE}$ and the one on the right hand side as $\mathcal{R}_{BDHE}$. We say that the Decision q-BDHE assumption holds in $\mathbb{G}$ if no polynomial-time algorithm has a non-negligible advantage in solving the problem.

Terminology for Binary Trees

We denote some terminology for complete binary tree. Let $\mathcal{L} = \{1, \ldots, n\}$ be the set of leaves. Let $\mathcal{X}$ be the set of node names in the tree via some systematic naming order. For a leaf $i \in \mathcal{L}$, let $\mathsf{Path}(i) \subset \mathcal{X}$ be the set of nodes on the path from node i to the root (including i and the root).
For $R \subseteq \mathcal{L}$, let $\mathsf{Cover}(R) \subset \mathcal{X}$ be defined as follows. First mark all the nodes in $\mathsf{Path}(i)$ if $i \in R$. Then $\mathsf{Cover}(R)$ is the set of all unmarked children of marked nodes. It can be shown to be the minimal set that contains no node in $\mathsf{Path}(i)$ if $i \in R$ but contains at least one node in $\mathsf{Path}(i)$ if $i \notin R$.

Lagrange Interpolation

For $i \in \mathbb{Z}$ and $S \subseteq \mathbb{Z}$, the Lagrange basis polynomial is defined as $\Delta_{i,S}(z) = \prod_{j \in S, j \neq i} \frac{z-j}{i-j}$. Let $f(z) \in \mathbb{Z}[z]$ be a d-th degree polynomial. If $|S| = d+1$, from a set of $d+1$ points $\{(i, f(i))\}_{i \in S}$, one can reconstruct $f(z)$ as

$$f(z) = \sum_{i \in S} f(i) \cdot \Delta_{i,S}(z).$$

In our scheme in Appendix B.3, we especially use the interpolation for a first degree polynomial. In particular, let $f(z)$ be a first degree polynomial, one can obtain $f(0)$ from two points $(i_1, f(i_1)), (i_2, f(i_2))$ where $i_1 \neq i_2$ by computing

$$f(0) = f(i_1)\frac{i_2}{i_2 - i_1} + f(i_2)\frac{i_1}{i_1 - i_2}.$$

B.2 More Details on Revocable Key DP-ABE

Correctness of $\mathsf{rkDPABE}$ is defined as follows:

Definition 16 (Correctness)**.** *A rkDP-ABE scheme is* correct *if* $\mathsf{Setup}(1^\kappa) \to (PK, MK)$ *then*

$$\mathsf{Decrypt}(\mathsf{Encrypt}(m, (\omega, \mathbb{S}), t, PK), (\omega, \mathbb{S}), (\mathbb{O}, \psi),$$
$$\mathsf{KeyGen}(\mathrm{ID}, (\mathbb{O}, \psi), \mathrm{MK}, \mathrm{PK}), \mathsf{KeyUpdate}(\mathrm{R}, \mathrm{t}, \mathrm{MK}, \mathrm{PK}), \mathrm{PK}) \to \mathrm{m},$$

for all t, all m in the message space $\mathcal{M}$, all $\omega \in \mathbb{O}$ and $\psi \in \mathbb{S}$.

Game 10 $\mathbf{Exp}_{\mathcal{A}}^{\text{IND-sHRSS}}[\mathbf{rk}\mathcal{DPABE}, 1^\kappa]$:

$((\omega^\star, \mathbb{S}^\star), t^\star) \leftarrow \mathcal{A}(1^\kappa)$
$(PK, MSK) \leftarrow \mathsf{Setup}(1^\kappa)$
$\tilde{R} \leftarrow \mathcal{A}(PK)$
$(m_0, m_1) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot,(\cdot,\cdot),MK,PK)\mathcal{O}^{\mathsf{KeyUpdate}}(\cdot,\cdot,MK,PK)}(PK)$, where $|m_0| = |m_1|$
$b \xleftarrow{\$} \{0,1\}$
$CT^\star \leftarrow \mathsf{Encrypt}(m_b, (\omega^\star, \mathbb{S}^\star), t^\star, PK)$
$b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot,(\cdot,\cdot),MK,PK)\mathcal{O}^{\mathsf{KeyUpdate}}(\cdot,\cdot,MK,PK)}(CT^\star, \tilde{R}, PK)$
if $b' = b$ **then**
  return 1
else
  return 0

Oracle Query 6 $\mathcal{O}^{\mathsf{KeyGen}}(\mathrm{ID}, (\mathbb{O}, \psi), \mathrm{MK}, \mathrm{PK})$:

if $\omega^\star \in \mathbb{O}$ and $\psi \in \mathbb{S}^\star$ **then**
  if $\mathrm{ID} \notin \tilde{\mathrm{R}}$ **then**
    return $\perp$
return $SK_{(\mathbb{O},\psi),\mathrm{ID}} \leftarrow \mathsf{KeyGen}(\mathrm{ID}, (\mathbb{O}, \psi), \mathrm{MK}, \mathrm{PK})$

Oracle Query 7 $\mathcal{O}^{\mathsf{KeyUpdate}}(R, t, MK, PK)$:

if $t = t^\star$ **then**
  if $\tilde{R} \not\subseteq R$ **then**
    return $\perp$
return $UK_{R,t} \leftarrow \mathsf{KeyUpdate}(R, t, MK, PK)$

Definition 17. *An rkDP-ABE scheme is secure in the selective-set security notion if all polynomial time adversaries have at most a negligible advantage in Game 10.*

B.3 The Construction

Our revocable DP-ABE scheme will be based on a combination of DP-ABE [7], which itself is a combination of [35] and [25], and an ABE scheme supporting revocation [5]. Both subjective and objective access structures are those which there exist linear secret sharing schemes that realize them. We represent a subjective access structure $\mathbb{S}$ by a LSSS which we denote by (M, ρ) and for an objective access structure $\mathbb{O}$ we use the LSSS representation denoted by (N, π).

Let $\mathcal{U}_s, \mathcal{U}_o$ be the universe of of subjective and objective attributes respectively. We define the universe set of identities $\mathcal{U}_{\mathrm{ID}}$ as the set of leaves in the complete binary tree $\mathcal{L} = \{1, \ldots, n\}$. Let us denote by m the maximum size of subjective attribute set to be assigned to a key, i.e. we restrict $|\psi| \leq m$. By n we denote the maximum size of objective attribute set allowed to be associated with a ciphertext, i.e. we restrict $|\omega| \leq n$. Furthermore we denote the maximum number of rows allowed in a subjective access structure matrix to be $l_{s,\mathbf{max}}$. Now let $m' = m + l_{s,\mathbf{max}} - 1$ and $n' = n - 1$. Let d be the maximum of $|\mathsf{Cover}(R)|$ for all $R \subseteq \mathcal{U}_{\mathrm{ID}}$.

$\mathsf{Setup}(\kappa)$**:** The algorithm first picks a random generator $g \in \mathbb{G}$ and random exponents $\gamma, \alpha \in \mathbb{Z}_p$. It then defines three functions $F_s \colon \mathbb{Z}_p \to \mathbb{G}$, $F_o \colon \mathbb{Z}_p \to \mathbb{G}$ and $P \colon \mathbb{Z}_p \to \mathbb{G}$ by first randomly choosing $h_0, \ldots, h_{m'}, q_1, \ldots, q_{n'}, u_1, \ldots, u_d$ and setting

$$F_s(x) = \prod_{j=0}^{m'} h_j^{x^j}, F_o(x) = \prod_{j=0}^{n'} q_j^{x^j}, P(x) = \prod_{j=0}^{d} u_j^{x^j}.$$

The algorithm assigns the public key as $PK = (g, e(g,g)^{\gamma}, g^{\alpha}, h_0, \ldots, h_{m'}, q_1, \ldots, q_{n'}, u_1, \ldots, u_d)$. For all node $x \in \mathcal{X}$ in the tree, it randomly chooses $a_x \in \mathbb{Z}_p$ and $r_x \in \mathbb{Z}_p$ for defining a first degree polynomial $f_x(z) = a_x z + \alpha r_x + \gamma$. The master key is $MK = (\gamma, \alpha, \{a_x\}, r_x)$.

$\mathsf{Encrypt}(t, (\omega, \mathbb{S}), m, PK)$**:** The encryption algorithm takes as input a LSSS access structure (M, ρ) for subjective policy and an objective attribute set $\omega \subset \mathcal{U}_o$. Let M be a $l_s \times k_s$ matrix. The algorithm now randomly chooses $s, y_2, \ldots, y_{k_s} \in \mathbb{Z}_p$ and sets $\boldsymbol{u} = (s, y_2, \ldots, y_{k_s})$. It calculates $\lambda_i = \boldsymbol{M_i} \cdot \boldsymbol{u}$ (for $i = 1, \ldots, l_s$), where $\boldsymbol{M_i}$ is the vector corresponding to the ith row of M. Furthermore the algorithm takes the present time attribute as an input. It then computes the ciphertext $CT = (C, C^{(1)}, \{C_k^{(2)}\}_{k \in \omega}, \{C_i^{(3)}\}_{i=1,\ldots,l_s}, C^{(4)})$, where

$$C = m \cdot (e(g,g,)^{\gamma})^s, C^{(1)} = g^s, C_k^{(2)} = F_o(k)^s,$$
$$C_i^{(3)} = g^{\alpha \lambda_i} F_s(\rho(i))^{-s}, C^{(4)} = P(t)^s.$$

$\mathsf{KeyGen}(PK, MK, (\mathbb{O}, \psi), \mathrm{ID})$**:** The key generation algorithm takes as input a LSSS access structure (N, π) for objective policy and a subjective attribute set $\psi \subset \mathcal{U}_s$. Let N be a $l_o \times k_o$ matrix and the algorithm takes identity $\mathrm{ID} \in \mathcal{U}$ which is a leaf in the binary tree. It then computes the key as follows.
For all $x \in \mathsf{Path}(\mathrm{ID})$, the algorithm first shares $f_x(1)$ with the LSSS (N, π) and it randomly chooses $z_{x,2}, \ldots, z_{x,k_o} \in \mathbb{Z}_p$ and sets $\boldsymbol{v_x} = (f_x(1), z_{x,2}, \ldots, z_{x,k_o})$. For $i = 1, \ldots, l_o$, it calculates the share $\sigma_{x,i} = \boldsymbol{N_i} \cdot \boldsymbol{v_x}$, where $\boldsymbol{N_i}$ is the vector corresponding to the ith row of N. Then the algorithm randomly chooses $r_{x,1}, \ldots, r_{x,l_o} \in \mathbb{Z}_p$ and $r_x \in \mathbb{Z}_p$ and outputs the private key as $SK_{(N,\pi),\mathrm{ID}} = ((D_{x,i}^{(1)}, D_{x,i}^{(2)})_{x \in \mathsf{Path}(\mathrm{ID}), \mathrm{i}=1,\ldots,\mathrm{l_o}}, (D, \{D_k^{(3)}\}_{k \in \psi})_{x \in \mathsf{Path}(\mathrm{ID})})$, where

$$D = g^{r_x}, D_{x,i}^{(1)} = g^{r_{x,i}},$$
$$D_{x,i}^{(2)} = g^{\sigma_{x,i}} F_o(\pi(i))^{r_{x,i}}, D_k^{(3)} = F_s(k)^{r_x}.$$

$\mathsf{KeyUpdate}(PK, MK, R, t)$**:** The algorithm first runs $\mathsf{Cover}(R)$ in order to find a minimal node set that covers $\mathcal{U} \setminus R$. For each $x \in \mathsf{Cover}(R)$, it randomly chooses $r_x \in \mathbb{Z}_p$ and sets the update keys as $UK(R,t) = (U_x^{(1)}, U_x^{(2)})_{x \in \mathsf{Cover}(R)}$, where

$$U_x^{(1)} = g^{f_x(t)} P(t)^{r_x}, U_x^{(2)} = g^{r_x}.$$

$\mathsf{Decrypt}(PK, CT, (\omega, \mathbb{S}), SK_{(N,\pi),\mathrm{ID}}, (\mathbb{O}, \psi), UK(R,t))$**:** The decryption algorithm takes as an input the ciphertetxt CT which contains a subjective access structure (M, ρ) and a set of objective attributes ω, and a decryption key $SK_{(N,\pi),\mathrm{ID}}$ which contains a set of subjective attributes ψ and an objective access structure (N, π). Suppose that the set ψ for subjective attribute satifies (M, ρ) and that the set ω satisfies (N, π) as well as $\mathrm{ID} \notin \mathrm{R}$ (so that decryption is possible). Let $I_s = \{i : \rho(i) \in \psi\}$ and $I_o = \{i : \pi(i) \in \omega\}$. The algorithm now constructs the respective set of reconstruction constants $\{(i, \mu_i)\}_{i \in I_s} = \mathsf{Recon}_{(M,\rho)}(\psi)$ and $\{(i, \nu_i)\}_{i \in I_o} = \mathsf{Recon}_{(N,\pi)}(\omega)$. Since $\mathrm{ID} \notin \mathrm{R}$, the algorithm also finds a node x such that $x \in \mathsf{Path}(\mathrm{ID}) \cap \mathsf{Cover}(\mathrm{R})$. Then it computes the following

$$C \cdot \frac{\prod_{i \in I_s} \left(e(C_i^{(3)}, D) \cdot e(C^{(1)}, D_{\rho(i)}^{(3)}) \right)^{\mu_i}}{\left(\prod_{j \in I_o} \left(\frac{e(D_{x,j}^{(2)}, C^{(1)})}{e(C_{x,\pi(j)}^{(2)}, D_{x,j}^{(1)})} \right)^{\nu_j} \right)^{\frac{t}{t-1}} \left(\frac{e(U_x^{(1)}, C^{(1)})}{e(C^{(4)}, U_x^{(2)})} \right)^{\frac{1}{1-t}}} = m.$$

Correctness.

We verify the correctness of the decryption as follows. Let us write the fraction in the decryption algorithm as C'/K and $K = (K')^{\frac{t}{t-1}}(K'')^{\frac{1}{1-t}}$. We first consider each part seperately. First we compute

$$\begin{aligned}
C' &= \prod_{i \in I_s} \left(e(C_i^{(3)}, D) \cdot e(C^{(1)}, D_{\rho(i)}^{(3)}) \right)^{\mu_i} \\
&= \prod_{i \in I_s} \left(e(g^{\alpha\lambda_i} F_s(\rho(i))^{-s}, g^{r_x}) \cdot e(g^s, F_s(\rho(i))^{r_x}) \right)^{\mu_i} \\
&= \prod_{i \in I_s} \left(e(g,g)^{\alpha\lambda_i r} \cdot e(g, F_s(\rho(i)))^{-r_x s} \cdot e(g, F_s(\rho(i)))^{r_x s} \right)^{\mu_i} \\
&= e(g,g)^{s r_x \alpha}.
\end{aligned}$$

The second equality follows from the construction, the third one form the properties of bilinear maps and the last equality follows from the set of reconstruction constants with $\sum_{i \in I_s} \mu_i \lambda_i = s$. In the next step we compute the following

$$\begin{aligned}
K' &= \prod_{j \in I_o} \left(\frac{e(D_{x,j}^{(2)}, C^{(1)})}{e(C_{x,\pi(j)}^{(2)}, D_{x,j}^{(1)})} \right)^{\nu_j} \\
&= \prod_{j \in I_o} \left(\frac{e(g^{\sigma_{x,j}} F_o(\pi(j))^{r_{x,j}}, g^s)}{e(F_o(\pi(j))^s, g^{r_{x,j}})} \right)^{\nu_j} \\
&= \prod_{j \in I_o} \left(\frac{e(g,g)^{\sigma_{x,j} s} \cdot e(g, F_o(\pi(j)))^{r_{x,j}, s}}{e(g, F_o(\pi(j)))^{r_{x,j}, s}} \right)^{\nu_j} \\
&= e(g,g)^{s f_x(1)}.
\end{aligned}$$

The second equality follows from the construction, the third one follows from the properties of bilinear maps and the last equation follows from the set of reconstruction constants with $\sum_{j \in I_o} \nu_j \sigma_{x,j} = a_x + \gamma + \alpha r_x$.
Now we can combine K' with the update key material to obtain

$$\begin{aligned}
K &= (K')^{\frac{t}{t-1}} \left(\frac{e(U_x^{(1)}, C^{(1)})}{e(C^{(4)}, U_x^{(2)})} \right)^{\frac{1}{1-t}} \\
&= (e(g,g)^{s f_x(1)})^{\frac{t}{t-1}} \left(\frac{e(g^{f_x(t)} P(t)^{r_x}, g^s)}{e(P(t)^s, g^{r_x})} \right)^{\frac{1}{1-t}} \\
&= (e(g,g)^{s f_x(1)})^{\frac{t}{t-1}} \left(\frac{e(g,g)^{f_x(t) s} \cdot e(g, P(t)^{r_x s})}{e(g, P(t)^{r_x s}} \right)^{\frac{1}{1-t}} \\
&= e(g,g)^{s(\gamma + \alpha r_x)},
\end{aligned}$$

which is indeed the Lagrange interpolation from two points $(1, f_x(1)), (1, f_x(t))$ to evaluate $f_x(0) = \alpha r_x + \gamma$.
Combining all of those steps we obtain

$$C \cdot \frac{e(g,g)^{s r_x \alpha}}{e(g,g)^{s(\gamma + \alpha r_x)}} = C \cdot \frac{e(g,g)^{s r_x \alpha}}{e(g,g)^{s\gamma} \cdot e(g,g)^{s \alpha r_x}} = m.$$

Sketch. Suppose there exist an adversary $\mathcal{A}$ that has an advantage ϵ in attacking the rkDP-ABE scheme. We build a simulator $\mathcal{B}$ that solves the Decision q-BDHE problem (Appendix B.1) in $\mathbb{G}$. The simulator $\mathcal{B}$ is given a random q-BDHE challenge $(g, h, \boldsymbol{y}_{g,a,q}, Z)$ where $\boldsymbol{y}_{g,a,q} = (g_1, \ldots, g_q, g_{q+2}, \ldots, g_{2q})$ as input and Z is either $e(g_{q+1}, h)$ or a random element in $\mathbb{G}_1$. Recall that $g_j = g^{a^j}$.

Initialization. The selective-set game begins with the adversary $\mathcal{A}$ choosing $(\omega^\star, (M^\star, \rho^\star))$, where $(M^\star, \rho^\star)$ is a target subjective access structure in the form of a LSSS matrix and as usual $\omega^\star$ is a target objective attribute set. Let the matrix $M^\star$ be of size $l_s^\star \times k_s^\star$, where $m + k_s^\star \leq q$ and w.l.o.g. we assume that $l_{s,\mathbf{max}}$ and $|\omega^\star| = n$. Furthermore due to the indirect mode ouf our scheme we have the auxiliary input $t^\star$.

Setup. The simulator $\mathcal{B}$ wants to generate PK. In order to do so $\mathcal{B}$ has to program the polynomials F_s, F_o and P (as in [5] and [6]) and chooses $\gamma \in \mathbb{Z}_p$ and implicitly sets $\gamma = \gamma' + \alpha^{q+1}$. We start with programming the function F_s. The simulator $\mathcal{B}$ defines $F_s(x) = g^{p(x)}$, where p is a polynomial in $\mathbb{Z}_p[x]$ of degree $m + l_s^\star - 1$ which is implicitly defined in the following way. It first chooses $k_s^\star + m + 1$ polynomial $p_0, \ldots, p_{k_s^\star + m}$ in $\mathbb{Z}_p[x]$ of degree $m + l_s^\star - 1$ in such a way that there exists an i where $x = \rho^\star(i)$ (there are exactly $l_s^\star$ values of such x, since $\rho^\star$ is injective). Now we set

$$p_j(x) = \begin{cases} M^\star_{i,j} & \text{for } j \in [1, k_s^\star] \\ 0 & \text{for } j \in [k_s^\star + 1, k_s^\star + m] \end{cases}$$

and p_0 is chosen completely at random. We can now write the coefficients in each polynomial as $p_j(x) = \sum_{i=0}^{m+l_s^\star - 1} p_{j,i} \cdot x^i$. It then defines

$$p(x) = \sum_{j=0}^{k_s^\star + m} p_j(x) a^j.$$

Set $h_i = \prod_{j=0}^{k_s^\star + m} g_j^{p_{j,i}}$ for $i \in [0, m + l_s^\star - 1]$. From combining these steps and the definition of F_s we obtain

$$F_s(x) = \prod_{i=0}^{m+l_s^\star - 1} h_i^{x^i} = g^{p(x)}.$$

The simulator then programs the next function F_o in the following way. $\mathcal{B}$ randomly picks a polynomial in $\mathbb{Z}_p[x]$ of degree $n-1$, $f'(x) = \sum_{j=0}^{n-1} f'_j x^j$. Next $\mathcal{B}$ defines $f(x) = \prod_{k \in \omega^\star}(x - k) = \sum_{j=0}^{n-1} f_j x^j$. We note that f_j's terms can be computed completely from $\omega^\star$. From this we can ensure that $f(x) = 0$ if and only if $x \in \omega^\star$. It then lets $q_j = g_q^{f_j} g^{f'_j}$ for $j = [0, n-1]$. Therefore we have

$$F_o(x) = \prod_{j=0}^{n-1} q_j^{(x^j)} = g_q^{f(x)} g^{f'(x)}.$$

Next the simulator defines

$$p(y) = y^{d-1} \cdot (y - t^\star) = \sum_{j=0}^{d} p_j y^j.$$

From this we can ensure that for all $t \in \mathcal{T}$, $p(t) = 0$ if and only if $t = t^\star$ and that for $x \in \mathcal{X}$, $p(x) \neq 0$ which follows from $\mathcal{T} \cap \mathcal{X} = \emptyset$. $\mathcal{B}$ then picks randomly a degree d polynomial in $\mathbb{Z}_p[x]$ as $\rho(x) = \sum_{j=0}^{d} \rho_j x^j$ and then lets $u_j = (g^a)^{p_j} g^{\rho_j}$ for $j = 0, \ldots, d$. Thus we have

$$P(x) = \prod_{j=0}^{d} u_j^{x^j} = (g^a)^{p(x)} g^{\rho(x)}.$$

It gives $\mathcal{A}$ the public key $PK = (g, e(g,g)^\gamma, g^\alpha, h_0, \ldots, h_{m'}, q_1, \ldots, q_{n'}, u_1, \ldots, u_d)$.

Query Phase 1. The adversary now makes requests for private keys corresponding to objective access structure and subjective attribute set pairs $((N, \pi), \psi)$ subject to the condition that ψ does not satisfy $M^\star$ *or* $\omega^\star$ does not satisfy N. Furthermore, since we deal with the semi-static query notion, at the beginning of this phase the adversary $\mathcal{A}$ has to announce $\tilde{R}$. Let $\mathcal{X}_{\tilde{R}} = \{x \in \mathsf{Path}(\mathrm{ID}) : \mathrm{ID} \in \tilde{\mathrm{R}}\}$. The simulator $\mathcal{B}$ must simulate answers to queries in the following 4 cases.

1. $\omega^\star$ does not satisfy N: $\mathcal{B}$ randomly choses $r_x \in \mathbb{Z}_p$. It then lets $D = g^{r_x}$ and for all $k \in \psi$ lets $D_k^{(3)} = F_s(k)^{r_x}$ as in the construction. Due to the condition that $\omega^\star$ does not satisfy N and Proposition 1, there must exist a vector $\boldsymbol{a} = (a_1, \ldots, a_{k_o}) \in \mathbb{Z}_p^{k_o}$ such that $a_1 = -1$ and that for all i where $\pi(i) \in \omega^\star$, it holds that $\boldsymbol{N_i} \cdot \boldsymbol{a} = 0$. The simulator now randomly chooses $z'_{x,2}, \ldots, z'_{x,k_o} \in \mathbb{Z}_p$ and $\boldsymbol{v'} = (0, z'_{x,2}, \ldots, z'_{x,k_o})$. It then implicitely defines a vector $\boldsymbol{v_x} = -(\gamma + \alpha r_x + a_x)\boldsymbol{a} + \boldsymbol{v'_x}$ which will be used for creating the share of $\gamma + \alpha r_x + a_x$ as in our construction. Now we consider the case that $\pi(i) \in \omega^\star$. Here we randomly choose $r_{x,i} \in \mathbb{Z}_p$ and compute $D_{x,i}^{(1)} = g^{r_{x,i}}$ and $D_{x,i}^{(2)} = g^{\boldsymbol{N_i} \cdot \boldsymbol{v'_x}} F_o(\pi(i))^{r_{x,i}} = g^{\boldsymbol{N_i} \cdot \boldsymbol{v_x}} F_o(\pi(i))^{r_{x,i}}$, where this holds because $\boldsymbol{N_i} \cdot \boldsymbol{a} = 0$. In case $\pi(i) \notin \omega^\star$ we have to observe some implicit relations between $\boldsymbol{N_i}$ and $\boldsymbol{v}$; we cannot compute as usual because the term α^{q+1} is missing. But we can use $F_o(\pi(i))^{r_{x,i}}$ to cancel out the unknown value and by redefining $r_{x,i} = r'_{x,i} + \frac{\alpha(\boldsymbol{N_i} \cdot \boldsymbol{a})}{f(\pi(i))}$ we are able to simulate the same keys for $D_{x,i}^{(1)}$ and $D_{x,i}^{(2)}$.

2. $\omega^\star$ does satisfy N: In this case we have to use that ψ does not satisfy $M^\star$. By Proposition 1 and with the same p_j as in the setup stage above, there exists a vector $(w_1, \ldots, w_{k_s^\star}) \in \mathbb{Z}_p^{k_s^\star}$ such that $w_1 = -1$ and for all $x \in \psi$ such that there exist i where $x = \rho^\star(i)$, we have $(p_1(x), \ldots, p_{k_s^\star}(x)) \cdot (w_1, \ldots, w_{k_s^\star}) = 0$. Next it also computes one possible solution of variables $w_{k_s^\star+1}, \ldots, w_{k_s^\star+m}$ for the system of $|\psi|$ equations. Since $|\psi| \leq m$ we have for all $x \in \psi$

$$(p_1(x), \ldots, p_{k_s^\star+m}(x)) \cdot (w_1, \ldots, w_{k_s^\star+m}) = 0.$$

The simulator $\mathcal{B}$ then randomly chooses $r'_x \in \mathbb{Z}_p$ and implicitly defines

$$r_x = r'_x + w_1 \cdot \alpha^q + w_2 \cdot \alpha^{q-1} + \cdots + w_{k_s^\star+m} \cdot \alpha^{q-(k_s^\star+m)+1}$$

by setting the key $D = g^{r'_x} \prod_{k=1}^{k_s^\star+m} (g_{q+1-k})^{w_k} = g^{r_x}$. From our definition of r and $\gamma = \gamma' + \alpha^{q+1}$, we have

$$\gamma + \alpha r_x + a_x = a_x + \gamma' + \alpha r'_x + w_2 \cdot \alpha^q + \cdots + w_{k_s^\star+m} \cdot \alpha^{q-(k_s^\star+m)+2}$$

where the α^{q+1} term in γ has canceled out. The simulator now randomly chooses $z_{x,2}, \ldots, z_{x,k_o} \in \mathbb{Z}_p$ and lets $\boldsymbol{v_x} = (\gamma + \alpha r_x + a_x, z_{x,2}, \ldots, z_{x,k_o})$ as in the construction. It

now randomly chooses $r_{x,1}, \ldots, r_{x,l_o} \in \mathbb{Z}_p$ and computes for $i = 1$ to l the key $D_{x,1}^{(1)} = g^{r_{x,i}}$. The other keys are computed in the following way. We have

$$D_{x,i}^{(2)} = \left(g^{a_x+\gamma'} \cdot g_1^{r'_x} \prod_{k=2}^{k_s^\star+m} (g_{q-k+2})^{w_k} \right)^{N_{i,1}} \cdot \prod_{j=2}^{k_o} g^{N_{i,j} z_j} F_o(\pi(i))^{r_{x,i}}$$

which can be computed since g_{q+1} is not contained and it follows that $D_{x,i}^{(2)} = g^{\boldsymbol{N_i} \cdot \boldsymbol{v_x}} \cdot F_o(\pi(i))^{r_i}$. The simulator then creates $D_k^{(3)}$ and since we have $(p_1(x), \ldots, p_{k_s^\star+m}(x)) \cdot (w_1, \ldots, w_{k_s^\star+m}) = 0$ for all $k \in \psi$ we have

$$\begin{aligned} D_k^{(3)} &= (g^{r_x})^{p_0(k)} \prod_{j=1}^{k_s^\star+m} \left(g_j^{r'_x} \prod_{k=1}^{k_s^\star+m} (g_{q+1-k+j})^{w_k} \right)^{p_j(k)} \\ &= (g^{r_x})^{p_0(k)} \prod_{j=1}^{k_s^\star+m} (g^{r_x})^{\alpha^j p_j(k)} = (g^{r_x})^{p(k)} = F_s(k)^{r_x}. \end{aligned}$$

3. $t = t^\star$ and $\tilde{R} \subseteq R$: To create $\left(U_x^{(1)}, U_x^{(1)}\right)_{x \in \mathsf{Cover}(R)}$, the simulator $\mathcal{B}$ chooses a random $r_x \in \mathbb{Z}_p$ for each $x \in \mathsf{Cover}(R)$ and computes $U_x^{(1)} = (g^{a'_x t^\star}) P(t^\star)^{r_x}$ and $U_x^{(2)} = g^{r_x}$. Both keys are valid since $\tilde{R} \subseteq R$ thus for all $x \in \mathsf{Cover}(R)$ we have $x \notin \mathcal{X}_{\tilde{R}}$. Thus$a'_x t^\star = f_x(t^\star)$ since for all x in the binary tree the adversary randomly chooses $a'_x \in \mathbb{Z}_p$ and pre-defines a_x by using a'_x in such a way that for $x \notin \mathcal{X}_{\tilde{R}}$ we have $f_x(t^\star) = a'_x t^\star$.

4. $t \neq t^\star$: Follows similary as in the previous case only that we have to differentiate between the cases that x is an element in $\mathsf{Cover}(R) \cap \mathcal{X}_{\tilde{R}}$ and $\mathsf{Cover}(R) \setminus \mathcal{X}_{\tilde{R}}$. Those update keys can be computed since $p(t)$ from setup does not equal 0 due to the fact that $t \in \mathcal{T}$, and we have that $p(t) = 0$ if and only if $t = t^\star$.

Challenge. The adversary $\mathcal{A}$ gives two messages m_0, m_1 to the simulator. The simulator flips a coin b and creates ciphertexts $C = m_b \cdot Z \cdot e(h, g^{\gamma'}), \hat{C} = h$, and for $x \in \omega^\star$ we write $C'_x = h^{f'(x)}$. We write $h = g^s$ for some unknow s. The simulator then chooses some random elements to share the secret s by redefining $\boldsymbol{v_x}$ and claim that if $Z = e(g_{q+1}, h)$ then the above ciphertexts are a valid challenge.

Query Phase 2. $\mathcal{B}$ performs exactly as in Query Phase 1.

Guess. $\mathcal{A}$ outputs $b' \in \{0, 1\}$ for its guess of b. If $b = b'$ then $\mathcal{B}$ outputs 1 which means that $Z = e(g_{q+1}, h)$. Else, it outputs 0 which means that Z is random from $\mathbb{G}_T$.
We see that if $(g, h, \boldsymbol{y}_{g,a,q}, Z)$ is sampled from $\mathcal{R}_{BDHE}$ then $\Pr[\mathcal{B}(g, h, \boldsymbol{y}_{g,a,q}, Z) = 0] = \frac{1}{2}$. On the other hand if $(g, h, \boldsymbol{y}_{g,a,q}, Z)$ is sampled from $\mathcal{P}_{BDHE}$ then we have $|\Pr[\mathcal{B}(g, h, \boldsymbol{y}_{g,a,q}, Z) = 0] - \frac{1}{2}| \geq \epsilon$. It follows that $\mathcal{B}$ has advantage at least ϵ in solving q-BDHE problem in $\mathbb{G}$. □

C Additonal Details for Access Control using DP-ABE

C.1 Access control on the delegator in PVC

Whilst the primary model we consider in Section 4 applies access control only to servers in order to maintain public delegability and verifiability, we note that in some situations it may

be desirable for the server to specify a policy over the functions or delegators that he will accept. For example, a server may only wish to receive a certain set of functions so that he can perform load management, or will only accept requests from delegators that have purchased a subscription for the service. In this brief section, we discuss some simple ways in which these additional policies can be incorporated into the above model.

Firstly, if delegators can be assumed to honestly describe the computations they outsource, then they can include additional descriptive attributes with the input data (recall that the input data shall be represented as a characteristic attribute set for a bitstring, and hence this is just a set union). When a server requests an evaluation key for a function F from the KDC, it can additionally specify an access control policy, $\mathbb{A}$. Since the functions we can outsource are Boolean functions, the KDC may issue a key for $F \wedge \mathbb{A}$. Thus, the computation can only succeed if the access control policy part of the key policy is also satisfied by the additional descriptive attributes provided by the client.

A slightly more compelling scenario is for the server to restrict requests to certain delegators only. A simple way to achieve this is for the KDC to define an additional set of attributes $\mathcal{U}_{\mathcal{D}}$ which describe delegators. During Setup the KDC normally publishes as part of the public parameters PP, a set of values corresponding to each attribute in the attribute universe. If, however, the KDC does not publish these terms for the subset $\mathcal{U}_{\mathcal{D}} \subseteq \mathcal{U}$, and instead issues them privately to only users described by each specific attribute, then only authorized delegators may use these attributes to create a ciphertext. As with the function policies above, the server may declare an additional policy when requesting an evaluation key, and then only those encoded inputs which contain valid attributes describing the user that satisfy this policy will be able to be evaluated.

Note that one downside to these methods is that a server must request a new evaluation key for each policy they wish to specify. We believe that it is likely that server policies will be based more on static factors such as descriptors of delegators and time rather than ephemeral contextual information about the computation in question (as with delegator policies). Thus, this may be a more reasonable restriction for computational servers. The same techniques can be applied in the reverse direction for VDC policies.

C.2 Security Model: Authorized Computation

The notion of Authorized Computation ensures that only a server that satisfies the additional authorization policy specified in the encoded input may perform a given computation and hence be rewarded for correct work. A server that does not satisfy this policy cannot produce a result that will cause the verification stages to accept the result (even if the result itself is correct). The challenger initializes a list L of authorization sets queried to the $\mathsf{Certify}$ oracle. The adversary loses if for any s queried, $s \in P$ where P is the challenge access control policy since the resulting key would lead to a trivial win.

Note that the proof is very similar to that for Public Verifiability in the HPVC setting. Instead of using dummy attributes and policies, we substitute for the authorization attributes and policies specified in the game.

D Authentication with DP-ABE

In this section we briefly demonstrate that DP-ABE can not only be used to provide confidentiality of messages or to prove validity of a computation, but also to allow a form of entity authentication and authenticated key agreement protocol. The basic idea is that two parties may be assigned a set of security labels for which they are authorized, and may decide (on a

Game 11 $\mathbf{Exp}_{\mathcal{A}}^{AuthComp}[\mathcal{HPVC}, \mathtt{mode}, F, 1^{\kappa}]$:

$L = \epsilon$
$(PP, MK) \leftarrow \mathsf{Setup}(1^{\kappa})$
$(x, P) \leftarrow \mathcal{A}^{\mathcal{O}}(SK_{\mathcal{A}}, PP)$;
for all $s \in L$ **do**
  if $s \in P$ **then**
    return 0
$(PK_F, L_F) \leftarrow \mathsf{FnInit}(F, MK, PP)$
$(\sigma_{x,P}, VK_{x,P}, RK_{x,P}) \leftarrow \mathsf{ProbGen}(\mathtt{PVC\text{-}AC}, (x, P), PK_F, PP)$
$\theta_{(F,s),(x,P)} \leftarrow \mathcal{A}^{\mathcal{O}}(\sigma_{x,P}, VK_{x,P}, EK_{(F,s),\mathcal{A}}, SK_{\mathcal{A}}, RK_{x,P}, PK_F, PP)$
$(RT_{(F,s),(x,P)}, \tau_{(F,s),(x,P)}) \leftarrow \mathsf{BVerif}(\theta_{(F,s),(x,P)}, VK_{(x,P)}, PP)$
if $(\tau_{(F,s),(x,P)} \neq (\perp, (\mathsf{reject}, \mathcal{A})))$ **then**
  return 1
else
  return 0

per-protocol run basis) a security policy determining the sets of security labels they require a communicating party to possess. Then, instead of directly running a (one-round) key agreement protocol, they encrypt their messages using their own security attributes and their chosen policy and exchange the resulting ciphertexts.

Now, each party can decrypt the received ciphertext if and only if both sets of attributes satisfy the policy dictated by the other party. If successful, both parties can complete the key agreement protocol to generate a shared, secret key to be used in further cryptographic primitives. Note that if either party does not satisfy the specified policy then neither will be able to decrypt. Thus, use of the resulting secret key is both proof that the user is indeed the party that was interacted with in the above protocol *and* that the user satisfies a security policy. This means that the secret key can act as a form of ticket providing authorization for further services without needing to look up the identity of the user in additional access control policies for example. This protocol is akin to those studied in [1, 21, 27, 32] for example where entities are not authenticated as individuals but rather as members of a group associated with certain attributes. We also note that the use of DP-ABE provides a form of implicit key confirmation. Since decryption succeeds if and only if *both* parties satisfy the specified policies, if one party successfully decrypts (i.e. the $\mathsf{Decrypt}$ algorithm does not output $\perp$), then they implicitly know that the other party could also decrypt successfully and hence they could both access the same key agreement messages[8]. This would not be the case if single-policy ABE were used since then one party may be able to decrypt and access the full key whilst the other could not.

As a simple example, Protocol 1 illustrates an extension using DP-ABE of a two-party key agreement protocol $\mathcal{KA} = (\mathsf{KA.Gen}, \mathsf{KA.Combine})$ where $\mathsf{KA.Gen}$ generates a message m_A if run by entity A, and $\mathsf{KA.Combine}$ takes two such messages and combines them to compute a shared secret key. Let the two entities be denoted A and B, and let each be assigned a set of security labels (attributes), X_A and X_B respectively, by a KDC. Each may determine a security policy $\mathbb{S}_A$ and $\mathbb{S}_B$ respectively which may be applied to particular protocol runs as determined by the entities themselves (based on the proposed use of the agreed key for example), and request a secret key for this policy and their assigned attributes from the KDC[9]. The two parties then encrypt their key agreement messages using this policy and their attributes and exchange ciphertexts. Decryption, in both cases, succeeds if and only if $X_A \in \mathbb{S}_B$ *and* $X_B \in \mathbb{S}_A$ – that is, both parties attributes satisfy the other's security policy.

By the indistinguishability security of the DP-ABE scheme, no information about the key

[8] A message integrate check would still be required by both parties however.

[9] Note that they may request many such keys for different policies ahead of time and select an appropriate one for each protocol run.

Protocol 1

A $\rightarrow$ **KDC:** $\mathbb{S}_A$
KDC$\rightarrow$ **A:** $SK_{(\mathbb{S}_A, X_A)} \leftarrow \mathsf{DPABE.KeyGen}(PP, MK, (\mathbb{S}_A, X_A))$
B $\rightarrow$ **KDC:** $\mathbb{S}_B$
KDC$\rightarrow$ **B:** $SK_{(\mathbb{S}_B, X_B)} \leftarrow \mathsf{DPABE.KeyGen}(PP, MK, (\mathbb{S}_B, X_B))$
A: $m_A \leftarrow \mathsf{KA.Gen}(1^\kappa)$
B: $m_B \leftarrow \mathsf{KA.Gen}(1^\kappa)$
A $\rightarrow$ **B:** $CT_A = \mathsf{DPABE.Encrypt}(PP, m_A, (\mathbb{S}_A, X_A))$
B $\rightarrow$ **A:** $CT_B = \mathsf{DPABE.Encrypt}(PP, m_B, (\mathbb{S}_B, X_B))$
A: $m_A \leftarrow \mathsf{DPABE.Decrypt}(PP, (\mathbb{S}_B, X_B), SK_{(\mathbb{S}_A, X_A)}, (\mathbb{S}_A, X_A), CT_B)$
B: $m_B \leftarrow \mathsf{DPABE.Decrypt}(PP, (\mathbb{S}_A, X_A), SK_{(\mathbb{S}_B, X_B)}, (\mathbb{S}_B, X_B), CT_A)$
A: SK $\leftarrow \mathsf{KA.Combine}(m_A, m_B)$
B: SK $\leftarrow \mathsf{KA.Combine}(m_A, m_B)$

Figure 3: Authenticated key exchange protocol using DP-ABE

agreement messages leaks unless *both* parties satisfy the policies – that is, $X_A \in \mathbb{S}_B$ *and* $X_B \in \mathbb{S}_A$ except with negligible probability. Thus, the parties agree on the same key if and only if they guess the other party's key agreement message (which happens with only negligible probability if p is large enough) or they are both authorized in relation to the other party's policy.

Since the DP-ABE mechanism is public-key, anybody may encrypt messages. Thus, a trusted authority (perhaps the KDC) could encrypt messages of the form in Protocol 1 with plaintexts specifying a key, which both (or neither) parties will be able to read and use if both parties request a decryption key for the specified policy and their own security attributes that satisfy it. This would provide a key distribution mechanism (with key escrow) only to authorized parties.

Finally, we observe that as well as authentication protocols, a similar mechanism could be used for any fair exchange of information protocol where each party sends (encrypted) data that should only be accessed by the other party if *both* parties are authorized according to a policy dictated by the other – thus, if either party is not authorized then neither learn any information about the data being exchanged. In the context of VC, it may be that one may access some additional data (the plaintext) only if both parties hold evaluation keys for functions that accept the other party's input data.